AD NUMBER

AD465805

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

**Distribution authorized to U.S. Gov't. agencies and their contractors;
Administrative/Operational Use; MAR 1965. Other requests shall be referred to Army Ballistic Research Laboratories, Aberdeen Proving Ground, MD 21005-5066.**

AUTHORITY

USABRL per ltr, 5 Nov 1965

BRL R 1273

BRL

REPORT
NO. 1273

THE FORAST PROGRAMMING LANGUAGE FOR ORDVAC AND BRLESC (REVISED)

By Lloyd W. Campbell
Glenn A. Beck

MARCH 1965

U. S. ARMY MATERIEL COMMAND
BALLISTIC RESEARCH LABORATORIES
ABERDEEN PROVING GROUND, MARYLAND

Destroy this report when it is no longer needed.
Do not return it to the originator.

B A L L I S T I C R E S E A R C H L A B O R A T O R I E S

REPORT NO. 1273

(Supersedes Report No. 1172)

MARCH 1965

THE FORAST PROGRAMMING LANGUAGE FOR ORDVAC AND BRLESC (REVISED)

Lloyd W. Campbell
Glenn A. Beck

RDT & E Project No. 1P014501A14B

A B E R D E E N P R O V I N G G R O U N D, M A R Y L A N D

B A L L I S T I C R E S E A R C H L A B O R A T O R I E S

REPORT NO. 1273

LWCampbell/GABeck
Aberdeen Proving Ground, Md.
March 1965

THE FORAST PROGRAMMING LANGUAGE FOR ORDVAC AND BRLESC (REVISED)

ABSTRACT

FORAST is a procedure oriented programming language designed for use on the ORDVAC and BRLESC computers at BRL. Although it was designed for professional programmers, FORAST contains sufficient simple concepts to make it usable by a novice or journeyman. It permits the use of arithmetic formulas, some English word statements, and each computer accepts its own symbolic or absolute machine language. The latter feature permits the professional programmer to use the full power of each computer.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

I. INTRODUCTION

FORAST is a programming language that is being used on the ORDVAC and BRLESC computers at BRL. It allows a programmer to write a computer program in a language that is closer to the English language and conventional mathematical notation than the numerical machine language. This simplifies the task of writing a program, makes it easier for another person to read, and allows the same program to be translated and run on either ORDVAC or BRLESC. While FORAST is a "problem-oriented" language, it is closer to a good machine language than most similar languages. FORAST also allows each machine to translate its own assembly language so that the professional programmer may use the full power of the computer in any program.

The following objectives influenced the design of FORAST:

1. Fast translation with "load and go" operation.
2. Allow full usage of machine's capabilities.
3. Generation of efficient object programs.
4. Translator must fit 4096 word memory.
5. Human readability.
6. Used primarily for mathematical problems.
7. Compatability between ORDVAC and BRLESC.

While many of the above objectives conflict with each other, an attempt has been made to achieve an optimum compromise of the conflicting objectives.

The following program will illustrate a simple FORAST program. It will read a pair of floating point numbers called X and Y, compute a number called Z that is the square root of the sum of the squares of X and Y, and will print the three numbers, X, Y and Z in a floating point form. Each line of coding (and hence each key-punched card) may contain either one or more machine orders in assembly language or one

or more arithmetic formulas or statements. (The $ symbol is used to separate two separate statements on the same line).

LOCATION	O.T.	STATEMENTS
1.1		READ (X)(Y)
		Z = SQRT (X**2 + Y**2)
		PRINT (X)(Y)(Z)$ GOTO (1.1)
	END	GOTO (1.1)

This example illustrates four types of statements allowed in FORAST. The READ statement allows numbers to be read from cards. The second statement is an arithmetic formula. On the left of the equality sign is the name of the quantity that is to be computed. The formula or arithmetic expression to the right of the equality sign expresses the arithmetic that the computer should do to compute this new arithmetic quantity. Note that X^2 is written X**2 since superscripts or subscripts cannot be key punched as such. SQRT is the standard name that is used to represent the square root function. The PRINT statement allows numerical results to be punched on cards. The GOTO statement allows control to be directed to another statement rather than to the next one. Statements are done in the sequence that they are written except when a GOTO statement is used to specify otherwise. Note that the first statement has been given a "location" name of 1.1 and hence GOTO (1.1) means to go to that statement.

The last card (physically) of every FORAST program must be an END card that contains a GOTO statement that determines the initial statement to be done in the running program. It is well to remember that all FORAST programs are done in two steps, first the entire symbolic program is translated into a numerical machine language program with assigned storage locations, and secondly, the program is run and the

desired computations performed and results produced.

The END card signals the completion of the translate phase. No separate symbol is needed to signal the end of the running program; the program runs until all data cards have been utilized. (An extra blank card at the end of the deck is required.) This is the normal way of stopping such programs at BRL. However, if no data cards are used in the final part of a program, a GOTO (N.PROB)% statement should be used. (See page 106). There is a HALT statement that may also be used, however it is better to use GOTO (N.PROB)% because this will cause the computer to stop at a standard "problem completed" halt.

II. CHARACTER SET

FORAST allows the use of the 26 capital letters, the 10 decimal digits, the decimal point, and prime (apostrophe) without any special significance attached to any one of these characters. These characters may be combined to make-up the names and locations of variables and instructions.

The following characters have special meanings and must not be used in names:

Card Punches	Symbol and Meaning
x	+ is addition
y	- is subtraction
x - 4 - 8	* is multiplication
0 - 1	/ is division
4 - 8	(is left parenthesis
y - 4 - 8	) is right parenthesis
3 - 8	= is equals
0 - 4 - 8	% is end of statement
0 - 3 - 8	, is used to indicate indexed addresses

Card Punches	Symbol and Meaning
x - 5 - 8	< is less than
0 - 5 - 8	> is greater than
	** is exponentiation
	¢¢ is end of card (rest of the card is ignored)
< = or = <	is less than or equal
> = or = >	is greater than or equal

The blank character is allowed and is ignored except when it is included in alphabetic information that is in a PRINT, PUNCH or ALFN statement. The symbol b may be used to indicate a blank column to the key punch operator.

The letters I and O must be written so as to be distinguishable from the numbers 1 and 0. It is suggested that the letter I be written with definite crossbars at top and bottom and the number one be written as a straight line. A script letter $\mathcal{O}$ that is also larger than zero is recommended. Some care must also be taken when writing S's and 5's, Z's and 2's, B's and 8's and all other characters. Programmers are urged to check the key punching of their programs.

III. NAMES AND ADDRESSES

The programmer may use symbolic names to represent the names of variables and the locations of instructions and constants. The FORAST translators translate these names into numerical machine memory addresses. Hence each symbolic name represents a memory cell. FORAST also allows the use of absolute numerical addresses.

Symbolic addresses may be chosen so that they consist of one or more characters with the following restrictions:

1. Must not contain any of the "special" characters. (See II)
2. Must not contain more than six characters unless the characters after the first six are not required for unique identification. (BLOC names on BRLESC may have as many as eight characters with some restrictions. See page 45)

3. Must contain at least one character that is not a decimal digit, for example, a decimal point or some alphabetic character.
4. The leading character must not be zero. A leading zero is used to indicate absolute sexadecimal addresses.
5. SELF can be used only to refer to the "location counter". (See page 40).
6. Certain names (SIN, COS, LOG, etc.) have been reserved as the names of subroutines and should not be used as the names of variables. (See pages 75-95 for the complete list of subroutine names.)

Some examples of symbolic names are X; TU; A3; 4J1; 1.1; SINA. The name "1.1" may be either a symbolic address or a floating point number. If it is written in an arithmetic expression, it is a number, otherwise it is just a symbolic name. However, if it is followed by a comma, then it would always be a symbolic name.

Absolute machine addresses may be written in either decimal or sexadecimal. Sexadecimal addresses must have at least one leading zero and decimal addresses must not have a leading zero. Sexadecimal addresses are never used as numbers because non-zero numbers must not have a leading zero unless the first non-zero character is a decimal point. The single character 0 (zero) will be a number if it appears where numbers are allowed. Decimal addresses will be used as numbers in arithmetic expressions unless they are followed by a comma. Commas are not allowed in numbers! Negative decimal addresses may be written and will be stored as 2's complement addresses. The characters K, S, N, J, F, L, are used for the sexadecimal characters ten, eleven, . . . , fifteen. For BRLESC, absolute index register addresses must lie in the range 1 to 63 in decimal or 01 to 03L in sexadecimal.

Indexed addresses may be written almost anywhere in FORAST. A comma is used to indicate that an address is indexed and also separates the primary address from the index address. An address may be indexed

by one and only one index register. The actual or effective address used at run time is the sum of the primary address and the contents of the index register. Indexing may be used with any type of an address, not just those that are defined as blocks or arrays. Indexing is in addition to, not in place of, constant "subscripts", thus, if B1 to B6 is a block (see BLOC page 44), it is permissible to write B3,J. Some examples of indexed addresses are A,1: B4,J: ,E: 42,I4: C,14: C,OF: ,OF. NOTE: A,-1 is not permissible. To get the same effect, first put -1 into K and then use A,K.

Decimal or sexadecimal increments may be written with any symbolic address. Thus a constant may be added or subtracted from a primary address at translate time. In arithmetic expressions and formulas (including the left side of formulas), the increment must be written after the index name and the index name and increment enclosed in parentheses. A,(J+1): B,(1-2): X4,(+3) are examples of addresses with increments as they must be written in arithmetic expressions and formulas. Note that the increment still may be used even if there is no index register, thus X4,(+3) is an address that is three more than the address of X4 and is not indexed. A sign must be used to distinguish an increment from an index register address. In any place other than arithmetic expressions, (arithmetic expressions are allowed only in arithmetic formulas and IF statements) the parentheses are not necessary and the increment may be written either after the primary address or after the index address. Thus A - 3: 4R3 + 6,II: X,J - 14 are legal addresses any place other than arithmetic expressions and formulas.

NOTE CAREFULLY: Symbols of the form A + 3 outside of an arithmetic expression refer to the address that is three greater than the address assigned to A. It is safe to use such an expression when referring to numbers since all numbers occupy one machine word. However, such an expression should not be used when referring to locations of statements unless the programmer is familiar with the amount of code that is generated by the statements. Thus, if A is a location, beware of "GOTO (A+3)%". The machine language of ORDVAC is very different from that of BRLESC and thus the number of words produced by the compiler from a particular statement may be quite different on the two machines. Note also that A1 + 1 is not necessarily the same address as A2. They are the same only if A1 and A2 are locations in a consecutive block that started at A or A1 and includes A2.

IV. ARITHMETIC EXPRESSIONS

Arithmetic expressions are allowed only in arithmetic formulas and IF statements. They are used to indicate the arithmetic operations that the computer should perform at run time and the special characters described in II are used to indicate the types and sequence of the desired operations. Arithmetic expressions are written much like they are in normal algebraic and mathematical usage. However, some special rules are necessary because everything must be written on one line as a consecutive string of characters so that it is key punchable. Thus superscripts, subscripts, and the normal over and under method of writing fractions are not allowed and require special rules.

The operations that are allowed are:

1. + and - ; addition and subtraction
2. * and / ; multiplication and division
3. ** ; exponentiation
4. Single-valued functions of one variable. (subroutines)

Parentheses may be used to group these operations into any desired sequence and are used in the same manner as they are in mathematics. In the absence of parentheses, the operations that are lower on the above list are performed before those that are higher on the list. Thus multiplication is done before addition, subroutines before division, exponentiation before subtraction, etc. Hence A + B/C**2 is the same as A + (B/(C**2)) and C**2 is the way C^2 is written. For successive operations that are on the same level, special rules apply when parentheses are omitted. Successive add or subtract operations are grouped from the left, thus V + W - T is the same as (V + W) - T. Successive multiply or divide operations are grouped from the right, thus A * R/B * S is the same as A * (R/(B * S)). Since multiplication and division are grouped from the right, parentheses are only required around any numerator or denominator that involves addition or subtraction and usually requires less parentheses than a left to right grouping. It also leads naturally to a more efficient one-address machine code.

Implied multiplication is allowed by writing a name adjacent to either a left or right parenthesis or by following a right parenthesis with a left one. Thus A * B may also be written as (A)B or A(B) or (A)(B). It is not incorrect to use a redundant * symbol where a multiplication is so implied. Note that AB is a single symbolic address and does not denote multiplication of A times B.

Subroutines having only one argument and one result may be used in arithmetic expressions by writing the standard name of the subroutine followed by its argument enclosed in parentheses. The argument may be any arithmetic expression and may use subroutines. See page 75 for the standard list of subroutines allowed in arithmetic expressions. LOG(A+B) ; ARCTAN(X + EXP(R-S)) are some examples of the use of subroutines in arithmetic expressions.

Successive exponentiations should always be grouped by using parentheses. Without parentheses, ORDVAC groups them from left to right and BRLESC groups them from right to left as they should be. The power of the exponentiation may be any integer or non-integer number or arithmetic expression. Small integer negative numbers may and should be written without being enclosed in parentheses, i.e. **-2; **-5. However "**+" is illegal. A quantity being raised to a non-integer power must always be positive because the logarithm of that number is used. A power of .5 causes the square root function to be used. Constant integer powers of fifteen or less are accomplished by in-line multiply orders plus one division order for negative integers.

Three types of arithmetic expressions are allowed, they are floating point, integer and fixed point fraction. However, not more than one type of arithmetic may be used in the same arithmetic expression. The type of arithmetic used in an expression is usually the "MODE" type. (See page 69). However, it may be changed for any one expression by writing "FLT(" or "FIX(" or "INT(" in front of any arithmetic formula or relational clause in an IF statement. Constant numbers written in arithmetic expressions are converted to the form required for the type of arithmetic being used in the expression. Fixed point fractional numbers must be less than one in absolute value and must have decimal points. A subroutine and an ENTER statement may be used to change variables from one type of number to another. The standard subroutines allowed in formulas all assume floating point arguments and hence must be used only in floating point expressions. However, ABS (absolute value) may be used with fractions and integers that are not stored in index registers. Index registers on BRLESC are not full words and will always seem to be positive to an ABS operation! Division involving a "negative" index register on BRLESC usually gives incorrect results and a negative integer product on ORDVAC will have an improper zero sign bit. The power of exponentiation

must be a positive constant integer of fifteen or less in integer or fractional expressions. (BRLESC does allow integers to be raised to any variable power or any constant power that does not have a fractional part of .5 exactly.)

Constant numbers written in arithmetic expressions must obey the following rules:

(1) Commas are not allowed.
(2) Exponents and scale factors are not allowed.
(3) Signs are not allowed except for a minus sign after **. ("Signs" will be used as add or subtract operators and the numbers will be stored as positive.)
(4) Leading zeros are allowed only if the first non-zero character is a decimal point or if the number is zero.
(5) Fixed point fractions must always contain a decimal point and be less than one (sixteen on BRLESC) in absolute value.

V. ARITHMETIC FORMULAS

The arithmetic formula is the type of statement that is used the most in writing FORAST programs. This statement has an arithmetic expression that is to be evaluated written to the right of an = character and the result is stored in the address specified on the left of the =. The result may be stored in as many as fifteen different places by specifying more than one address and having more than one = symbol to the left of the arithmetic expression. Thus X = Y = A means to take the quantity named A and store it in the memory locations called X and Y. Note that the arithmetic expression may be just a single variable or number! The following examples illustrate some arithmetic formulas:

```
Y = 0 %                    FLT(Q = A = B** - 2%
X = X + 3%                 X,I = Y,I + SIN(A+B,I)/C,J*E
X = X, (+3) + 3%           R,(J+2) = T1 ** 2 + S ** COS(X3-X2)
FIX(X2,J = -X1 %           (Note: the index register J, like any
                            other variable, is set only at run time)
```

The formula X = X + 3 shows that this type of statement is not an equation that is to be solved. This example means that the value of X should be increased by three and stored back into the same memory location.

Parentheses may be used to group operations in the usual manner. They may be omitted at either end of the arithmetic part of the formula because all right parentheses that are not closed on the left will automatically be closed at the = symbol and all left parentheses that are not closed on the right will automatically be closed at the $ at the end of the formula. The number and amount of nesting of parentheses is practically unlimited. The amount of nesting is limited only by the fact that the translators can save only 30 operations that have been encountered by the left to right scan but not coded because of some right to left grouping, either by parentheses or by a succession of multiply and divide operations. Nesting that cause operations to be grouped from the left is essentially unlimited.

The operations of + and - may be used as unary operations only at the beginning of a formula or after a left parenthesis. The - symbol may also be used as a sign after ** if it is followed by a constant number. This is the only exception to the rule that two operation symbols (+ - */) must not be written adjacent to each other.

VI. ENGLISH WORD STATEMENTS

FORAST allows the use of a few English words to instruct the computer to do certain operations. There are eleven of these statements, each of which begins with a special English word that determines the type of statement. All of the statements may be interspersed among arithmetic formulas. Arithmetic expressions are not allowed in any of these

statements except in the relational clauses of the IF statement. All other statements use only addresses which may have an increment as described in III. The list of permitted English word statements is:

GOTO: SET: SETEA: INC: COUNT: IF: CLEAR: MOVE: ENTER: READ and PRINT or PUNCH: HALT

1. GOTO

   General Form: GOTO (Location to go to)%

The words GO TO (or GOTO) may be used to tell the computer to go to a location of a statement that does not follow the statement that has just been done. Statements are normally done in the same sequence they are written in, however this GOTO statement (and a few other statements) may be used to change that sequence. The location of the statement to be done next is enclosed in parentheses. Some examples are

```
GOTO(BOX2)%          GOTO(START)%
GOTO(4.2)%           GOTO( ,E)%
```

Note that the address may be indexed and the last example illustrates how "remote connections" can be handled in FORAST. Since an index register can hold an entire address, not merely the customary increment to an address, the address to go to may be whatever address has been last set into an index register (by a SET statement) in the running program. This idea of using index registers for "address substitution" may of course be used in any type of statement that allows indexed addresses. Note that the addition of the contents of an index register to the primary address is still performed and while the primary address is usually blank (zero) in an indexed GOTO statement, it does not have to be blank. Thus 3,V means to add 3 to the contents of V to get the address used and 3,V is the same as ,(V+3).

Since ORDVAC (See [1]) has both left and right orders (2 orders per word), the GOTO will go to the proper side of the word if the address is not indexed. Since the side it goes to is determined only by the primary address, caution must be used with indexed addresses. The effective address of all indexed addresses should be the location of a left order. This can be accomplished by using location names that begin with a letter other than R for all locations that are transferred to by an indexed GOTO. (See VIII page 38.)

a. Computed GOTO

General Form: GOTO, index add.(Loc. to go to when index = 1)Loc to go to when index = 2) . . . . . .%

This statement will cause a program to go to different places depending on the contents of an index register. When the contents of the index register is an integer i, the ith location name on the list of location names in this statement will be used as the location of the next statement that is to be done. For example;

GOTO,I(A)B),K%

will do a GOTO(A) statement when I = 1, and a GOTO(B) statement when I = 2 and a GOTO(,K) statement when I = 3.

A decimal increment on the index address is allowed when it is enclosed in parentheses. For example;

GOTO,(J1 - 1)(L1)L2)%

will cause a GOTO(L1) when J1 = 2 and a GOTO(L2) when J1 = 3. When an increment is used, the "extra" left parenthesis is required before the first location name on the list.

For ORDVAC, each location on the list may go to either a left order or a right order; they don't all have to go to the same side.

If by error, the value of the integer is zero, the computer will cycle on one jump order. If the integer is too large, it will go on to the program at some point below this statement.

2. SET

Form: SET (Index add. = add. to put in index register). . . . .

This statement should be used to set index registers to a constant value. It cannot be used to set floating point numbers. The value of the address itself (not the contents of) written on the right of the = is put into the index register specified. If this address is a decimal or sexadecimal base number, then that number is put into the index register. If this address is a symbolic name, then the address that has been assigned to that name is put into the index register. Hence this statement allows index registers to be set to addresses that have been assigned to symbolic names of variables or locations. Indexed addresses are not allowed (See 3. below) but a decimal increment is allowed when the address is symbolic. Some examples of SET statements are:

SET(I=0)J=2)A=A2% SET(B=0KN4)% ;

SET(V=-3)(N=+2)GOTO(6.2)%

Note that any number of index registers may be set with one SET statement with each one separated from the previous one by a right parenthesis or a right and a left parenthesis. A GOTO statement may be included at the end of a SET statement without a % in between.

3. SETEA

General form: SETEA (Index add. = Indexable address) ....

This statement is the same as the SET statement except that the address on the right of the = is indexable and GOTO is not allowed as part of this statement. Some examples of SETEA statements are

```
SETEA(I=A,J)%
SETEA(K=B,M+2) P = 14,117)%
```

This statement sets the effective address into the index register specified.

4. INC

General form: INC (Index add. = Index add. ± amount of increase or decrease) ....

This statement should be used to increase index registers by a constant amount. The index register name that is written after the "=" (the same name should appear on the left of the "=") is increased by the address itself (not the contents of) that is written after the first + or - sign that appears to the right of the "=". The amount of increase may be symbolic and may have an increment but cannot be indexed. A minus sign may be used to indicate a decrease only if the amount is not symbolic. Some examples of the INC statements are

```
INC(I=I+1)% INC(J=J-2)%
INC(S=S+03K)V=V+10)A=A+N-1 %
INC(R=R+41)GOTO( ,E)%
```

This statement may be used to increase (or decrease) any number of registers. A GOTO statement may be written at the end of an INC statement without a % in between.

5. COUNT

General Form: COUNT (max. count) IN (Index Reg.) GOTO (Loc. for repeating loop)

This statement can be used to count the number of times a loop is done and to also increase one index register. This index register (specified after IN) is increased by one (or another amount if it is specified) until it is greater than or equal to the maximum count specified after COUNT. (A symbolic "max. count" address will be used as an index register whether a comma precedes it or not). The increase occurs before the comparison! If the index register (specified after IN) is set to zero at the beginning of the loop and increased (by one only)in a COUNT statement at the end of the loop, then max. count is the total number of times the loop is to be done.

The amount to increase the index register may be specified in three different ways. It may be written after IN by writing the same type of formula as allowed in the INC statement, i.e. (I = I + 2) would cause I to be increased by two. It may also be written by inserting "BY (increase)" between the max. count and IN as shown in the third example below or it may be included inside the max. count parentheses by the form of (max. count/increase) as shown in the fourth example below. If no increase is specified, it will be used as one. If the increase is n and $n \geq 1$, then the max. count specified must be m times n in order that the loop be done m times. Thus COUNT (24/2)... actually means the loop will be done 12 times (assuming the index started at zero).

The max. count and the increase may be either absolute decimal or sexadecimal integers, either explicit integers or the contents, at run time, of index registers. In order to preserve the symmetry of these two integer representations, symbolic names of index registers may be written without the usual preceding comma. Thus COUNT (J/I-1)IN(K)... means to use the contents of index register J as the max. count and to increase the index register K by one less than the contents of I. If K starts at zero the loop would be done $\frac{J}{I-1}$ times. (If $\frac{J}{I-1}$ is not

an exact integer, the result appears to the machine to be rounded up to the nearest integer; thus, if J = 69 and I = 3 then J/I-1 would cause the loop to be done 35 times.) Note that arithmetic expressions are not allowed anywhere in a COUNT statement but that a constant increment or decrement may be used with any address. Note also that the address after GOTO (this GOTO may be replaced by any other English word) is the location to jump to when the index register has not reached its upper limit. This will usually be the location of the beginning of the loop since the COUNT statement will usually be the last statement in the loop. The index register being increased is available within the loop for indexing and the final increased value is available when the max. count has been reached and control passes to the next statement. Some examples of COUNT statement are

```
COUNT(20)IN(J2)GOTO(BOX 3)%
COUNT(5)IN(K=K+1)GOTO(LOC 6)%
COUNT(N+4)BY(2)IN(R)GOTO(17.1)%
COUNT(33/3)IN(I6)GO BACKTO(3.2)%
```

There is no inherent nesting limit of COUNT statements, the only limit of nesting is the number of index registers available (54 on BRLESC and about 3000 on ORDVAC). There are no restrictions on transferring into or out of the loops controlled by COUNT statements.

On BRLESC only, it is possible to omit the "IN (index reg.)" part of the COUNT statement if the increase amount is a constant number. Such a statement counts by itself and resets itself to zero when the limit is reached; however, the loop must not be left by some other statement.

6. IF

General Form: IF (ce) AND or OR (ce) AND or OR (ce)..... GOTO(Loc.)% where ce is any conditional expression that has the form (AE relation AE relation AE .....) where AE is any arithmetic expression and the relation is $<, \leq, >, \geq$ or =. The general form of "AE) IS $\pm$" is also permitted for any conditional expression.

This statement allows a conditional transfer of control to another statement. It goes to the location specified after GOTO whenever the statement is "true". If the statement is "false", control goes to the next statement. The AND condition always has precedence over OR and this cannot be changed by using parentheses (this means that the conditional expressions on both sides of any AND are grouped together). However, any desired grouping of AND and OR conditions can be obtained by writing enough IF statements and doing them in the proper sequence.

Each conditional expression may be preceded by any one or more of the following names that apply only to the next one conditional expression:

-NOT ; Negate the meaning of the following conditional expression. (Negate the relations and change the implied "and" condition of several relations to an "or" condition.) However, AND-NOT should not be used before a conditional expression that contains more than one relation. (The present FORAST translators will only negate the relations when this is used and will not change the implied "and" condition to "or".)

-FLT ; Use floating point arithmetic to check the truth of the next conditional expression.

-FIX ; Use fixed fractional arithmetic to check the truth of the next conditional expression.

-INT ; Use integer arithmetic to check the truth of the next conditional expression.

-ABS ; This cannot be used when the = relation is involved in the next conditional expression. For inequality relations, the absolute values of both arithmetic expressions are used to check the truth of all relations in the next conditional expression.

-IF ; Allowed so OR-IF or AND-IF may be written. It is also true that OR IF and AND IF (with or without the space) may be written instead of OR and AND.

MODE arithmetic (see page 69) is used to check the truth of any conditional expression that is not preceded by -FLT -FIX or -INT.

When the relation is =, a tolerance may be specified and the conditional expression may have the general form of "AE = AE) WITHIN(AE)" where AE is any arithmetic expression. Only one = relation is allowed before the WITHIN and the AE after WITHIN is the tolerance. The equality relation is considered to be true when the absolute value of the difference of the two quantities is less than or equal to the absolute value of the tolerance.

The GOTO portion of the IF statement may precede the IF or may appear after any conditional expression. If the GOTO is at the beginning, the conditional expressions are tested from left to right in the running code. If the GOTO is at the end of the IF statement, the conditional expressions are tested from right to left.

The following names should not be used as the names of arithmetic quantities in IF statements: GOTO: OR: AND: IS: WITHIN: ORIF:

ANDIF. (Actually these names may be used except immediately following a right parenthesis.)

Some examples of IF statements are:

```
IF(Y=16)OR(X+AL)IS+GOTO(64.7)
IF(A+B< R**2)AND(X > Y > 0)GOTO(LOC 3)
IF(Q=T-SIN(V/A))WITHIN(.001)GOTO(4.1)%
IF-INT(I=3)AND-INT-NOT(J=1)GOTO(DONE)%
IF-NOT(A=B=C)OR-ABS-NOT(X > = C3(GOTO( ,T)%
GOTO(WRONG)IF(X3-X1 < X3-X4 < =1)%
```

In the following two examples, the statements on the same line are equivalent:

```
IF(X > Y > 1)GOTO(A)% IF(X > Y)AND(Y > 1)GOTO(A)
IF-NOT(X > Y > 1)GOTO(A)% IF(X < =Y)OR(Y < =1)GOTO(A)
```

7. CLEAR

General Form: CLEAR (count) NOS.AT (initial address)%

This statement may be used to clear a group of uniformly spaced memory cells to zero. (Fl. pt., fixed fraction, and integer zeros are all identical.) The count is the number of cells to clear (or some multiple of it if the count increment is not one). The count is written in an index register and used exactly like the "max. count" in the COUNT statement. CLEAR(I/3) does not mean to operate on every third location, as one might expect, it simply means the contents of the desired index register I happens to be 3 times the number of the number of cells that are to be cleared.

The initial address is indexable and is the address of the first cell to be cleared. Consecutive cells are cleared unless a different amount of address advance is specified by writing it after a / after the initial address. Counting by more than one may be done by writing a larger counting increment after a / after the "count". If the counting

and address advancing should use the same increment, it may be written in parentheses just before %. If any of the count or address advance increments are symbolic, it uses the contents of that cell and assumes that the cell is an index register containing an integer number at run time.

The CLEAR statement always clears at least one cell even if the count is zero. If three or fewer cells are to be cleared it is more efficient to write arithmetic formulas instead of a CLEAR statement, e.g. an arithmetic formula of Y = 0 means to clear cell Y.

Some examples of CLEAR statements are:

```
CLEAR(20)NOS.AT(A)%
CLEAR(I/2)NOS.AT(X2)%
CLEAR(N+3)NOS.AT(R,J+1/4)%
CLEAR(K-3)NOS.AT(B1)(3)%
```

Note that the I/2 count in the second example actually means to clear I/2 cells, i.e. the / symbol here actually can be interpreted as indicating integer division with the nearest larger integer being used for inexact quotients. In the third example, every fourth cell is cleared to zero, until (N+3) cells have been cleared or 4(N+3) cells have been "passed over."

8. MOVE

General Form: MOVE (count) NOS.FROM (add.) TO (add.)%

This statement may be used to move the contents of a uniformly spaced group of memory cells to a different group of uniformly spaced memory cells. The count is exactly the same as was defined for the CLEAR statement. The initial addresses of each group of cells are written as shown above and each may be followed by a / and an address

advance increment that may be either positive or negative or zero. If it is symbolic, then it is assumed to be an index register and its contents are used as the advance. Only one symbolic name is allowed in each advance increment. Any count or advance increment not specified will be 1. If the increment for counting and advancing both addresses should be the same, it may be enclosed in parentheses and inserted just before %. At least one number is always moved, even if the count is zero. If three or fewer numbers are to be moved, it is more efficient to write arithmetic formulas to do the moving, e.g. an arithmetic formula of A = B% means to move the quantity from B into A.

Normally the initial address of the two groups of cells are used and positive advance increments are used. However, if the initial address of the "to" group of cells is the same as any one of the"from" group of cells, then the moving must be done "backwards" so that all cells get moved before they are moved into. In such a situation, end addresses of each group of cells must be specified in the MOVE statement and negative advance increments must also be specified. (See the last example below.)

Some examples of MOVE statements are:

```
MOVE(144)NOS.FROM(A1)TO(B1)%
MOVE(N+4/4)NOS.FROM(Q+2)TO(Q)%
MOVE(J)NOS.FROM(X,M/3)TO(Y,N)%
MOVE(R-2)NOS.FROM(C2/0)TO(V1/K+1)%
MOVE(600)NOS.FROM(B600/-1)TO(B700/-1)
```

9. ENTER

General Form: ENTER (subroutine name) (add.)....(add.)%

This statement allows the use of subroutines that do not conform to the one argument and one result type that are allowed in arithmetic formulas. (See pages 75-95 for the entire list of standard subroutines that are included in the present FORAST translators.) The subroutines allowed in formulas (except ABS) may also be entered with an ENTER statement. The list of addresses following the subroutine name are the addresses of the arguments and results and the number and meaning of these addresses varies with the subroutine being entered. The subroutine name address is not indexable (ORDVAC restriction) but most subroutines allow any of the other addresses to be indexed. An address specified for an argument or a result is usually the memory location that contains the argument or will contain the result. However some subroutines use some addresses (which are necessarily integers) as being an integer argument. This is done for dimensions of matrices, number of equations, number of points, etc. where the argument is often an integer constant and hence can be written as an address. If such an address is variable, then the integer argument must be stored in an index register and the address written with a comma in front of the index register name so that the effective address is the desired integer argument. Small letters are used in the list of subroutines (pages 75-95) to denote the addresses that are integer arguments.

Constant numbers may be written instead of the address of an argument only if preceded by an *. The type of number may be determined by inserting F,X or I between the * and the number. In the absence of F,X, or I, the number will be converted to the MODE type of arithmetic. (See DEC, page 56 for the rules for writing decimal constants.)

The ENTER statement is not restricted to entering standard subroutines, it may be used to enter any sequence of statements or machine instructions that provide for using the string of addresses and returning

to the statement that follows the ENTER statement when the "subroutine" is finished.

Some examples of ENTER statements are:

```
ENTER(SINCOS)X)SINX)COSX %
ENTER(SINCOS)(*2.7)SIN 2.7)(COS 2.7)%
ENTER(ARCTAN)V,I-1)ATV)%
ENTER(MAT.MP)A1)B1)C1)3),I+1)6 %
ENTER(PRINT BLANK)%
```

10. READ and PRINT or PUNCH

General Forms:

```
READ(add.)(add.).....%
READ(count)NOS.AT(add./increment)%
PRINT-FORMAT(format add./subgroup)-(add.)...(add.)%
PRINT(add)...< string of characters > .... %
```

The READ statement allows decimal numbers to be read from cards (or tape on BRLESC) and the PRINT (or PUNCH) statement allows decimal numbers and/or alphanumeric characters to be punched on cards (or, on BRLESC, to be put on tape). There is no difference between PRINT and PUNCH, the type of BRLESC output depends upon the setting of a console switch and the use of SET.TO as a statement or subroutine.

The addresses of the quantities to be read or printed may be specified as either a list of single addresses or by stating the total number of numbers (count), the initial address, and the address increment if it is other than one. The "count" must always be separated from the address by the use of "NOS.AT" and the "count" is always the total number of numbers actually read or printed (or punched) regardless of

whether the address increment is one or not. The count and address increment are integer numbers and a symbolic address will be used as an index register whether a comma precedes it or not.

The entire list of quantities involved in any one READ or PRINT or PUNCH statement may be any combination of single addresses and "NOS.AT" clauses. PRINT or PUNCH statements may also contain a string of alphanumeric characters that will be punched in addition to any numbers that are punched. The character < must precede the string and the character > ends the string. Any character except > may be used in a string and blank characters within a string are not ignored. A string of characters cannot be continued (by using CONT, see page 52) from one program card onto another. If a string is too long for one card, it must be written as two or more shorter strings with each one completely contained on a card. A string of all blank characters may be indicated by just writing "n >" where n is a decimal or sexadecimal number of blank columns to be inserted in the output. The n must be preceded by "(" if it appears first in the PRINT or PUNCH statement. For example: PRINT (7 >(A)21 >B% would skip 7 columns, print the number called A, skip 21 columns and print the number called B.

The type and length of decimal numbers read or punched is controlled by a format word. If no format is specified, then a standard format that allows six numbers of twelve columns each on each card is used. The input numbers may be either floating decimal with an exponent or with a decimal point punched (or both) and are stored as floating point numbers. (See FORM, T = 10 page 63). The standard output format assumes floating point or integer numbers and will print six numbers of twelve columns each. The floating point numbers will have exponents and an assumed (not punched) decimal point to the left of the coefficient. (See FORM, T = 9 page 63). (The standard floating point format for each number is sign and eight digit coefficient with sign and two digit exponent.)

A non-standard format may be specified in any READ or PRINT (or PUNCH) statement as illustrated in the third general form above. To

specify a format, PRINT (or READ or PUNCH) must be followed by a dash (minus sign) and the next name enclosed in parentheses is the name of the first format word to be used. It is suggested that the word FORMAT be written between the dash and format name. The format address may be followed by a / and a subgroup integer number. The subgroup is the number of numbers that are to be punched or read on one card (or one group of cards). Whenever the subgroup number of numbers has been read or punched, a new card is started and the format is started from the beginning. A zero or omitted subgroup means that there isn't any subgroup. The format address and the subgroup are not indexable (ORDVAC restriction). (See FORM, page 59 for information on storing format words.)

The name "NOS.AT" must not be used as the name of a number to be read or punched. Note that NOS.AT may be followed by a left parenthesis.

The standard minus sign used on input and output numbers is the y(or 12) punch. For input, any number that is not negative is positive and the standard output plus sign is a blank column. However the meaning of the sign punches may be changed by using the SETMSI, SETMSO, and SETPSO subroutines in an ENTER statement. (See page 78). Signs normally occupy a column by themselves and are said to be "single punched." However they may be "double punched" (punched above the leading digit of the number by using SETDPI (for input) or SETDPO (for output) in an ENTER statement. (See page 78). Note that double punched signs cannot be printed on the hi-speed printer, it would print some letter instead of a sign and a digit.

A blank card is used as a standard sentinel card for READ statements and reading will stop whenever a completely blank card is

read except when the blank card is the first card read by a READ statement. Thus a READ statement may be written to read a large amount of data and the actual amount of data stored may be controlled by inserting a blank card at the end of the data. (The maximum amount of numbers that may be specified is 16383 but ORDVAC uses the amount modulo 4096.) If it is desired that a READ statement should read no data, it is necessary to insert <u>two</u> blank cards because the first card is ignored if it is blank. The letter S (or the word STOP) punched in place of a number on an input card also stops the READ statement from reading in the same manner as a blank card does. (The S must not be punched in the sign column for ORDVAC.) If a field is punched with the letter X in any column except the sign column, no number is stored from this field. The next number will store in the same place the previous field would have stored. An "X field" will be counted in the subgroup count (if there is one) but not in the total number of numbers that is left in index 9 and the next format type is used for the next field. Note that an "X field" is a way of removing a number from the middle of a group of numbers without repunching them. It is not a way of not storing a number in an address specified in the READ statement.

Every READ or PRINT or PUNCH statement begins a new card. A new card is started within a statement only when the format or subgroup indicates that a new card should be started.

After a READ statement, the number of cards (not counting blank cards) read by that statement is always left in index 8 and the number of numbers stored is left in index 9 as integer numbers.

The strings of alphanumeric characters allowed in PRINT or PUNCH statements are entirely extra and are inserted on the card wherever they occur in the statement. If the string occurs at the same place a format word indicates a skip, start new card, etc., the string will be punched <u>before</u> the format action occurs.

The format does not need to include anything extra to print the alphanumeric characters nor is any part of the format word used or skipped while the string of characters is being printed. A number printed after a string of characters begins in whatever column follows the string on the card.

The symbols < and > may be used without parentheses between them and the addresses of a number. As usual, the left parenthesis preceding an address is optional after a right parenthesis or after >. The dash (minus sign) must always be used both before and after a FORMAT specification.

Some examples of READ and PRINT or PUNCH statements are:

```
READ(X)(Y)(Z)% READ(24)NOS.AT(A1)%
READ-FORMAT(F4)-(S)(T)(16)NOS.AT(B1,1)%
READ(U)(V)4)NOS.AT(X/2)(A)B)J+2)NOS.AT(R1/I-1)%
PRINT(X)(Y)(Z)% PRINT< X IS TOO BIG> (X)%
PRINT-FORMAT(QT/3)-(K)NOS.AT(A1)(OKS)NOS.AT(M1,1)%
PUNCH< X =>(X) <Y =>(Y) <Z =>Z%
PUNCH(4 >< HEADING > 6 >< RANGE > 5 >< HEIGHT > %
```

11. HALT
General Form: HALT (Display address)%

This statement causes the computer to stop running. The address is optional, but if it is used, it will be displayed in the halt order. (It will be in the first address of a BRLESC halt order.) If the computer is re-initiated, it will continue with the next instruction or statement. If a problem is done running or can not run further for some reason, a GOTO(N.PROB)% statement should be used instead of a HALT statement.

Examples: HALT % HALT(3) % HALT(ONO)%

VII. PROGRAM CARD FORMAT

FORAST program cards are divided into four fields as follows:

Columns	Use
1 - 6	Location field.
7 - 10	Order Type field.
11 - 76	Formula and Statement field.
77 - 80	Identification.

The location field (cols. 1 - 6) may be used to assign a name to the first statement or constant that appears on the card. (See VIII page 38).

The order type field (cols. 7 - 10) is used for the "pseudo order types" that provide translation information and may be used for the order type of assembly orders. The order type field determines how the rest of the card is interpreted and is to be left blank when the card contains arithmetic formulas and/or English word statements.

The formula and statement field (cols. 11 - 76) is primarily used for arithmetic formulas and English word statements. It may also be used for assembly orders, numbers, translation information, comments, etc. The meaning of this field is controlled by the order type field. If this formula and statement field is not long enough, it may be continued onto the next card by using CONT in the order type field of the next card. (See CONT page 52). This field may be terminated before column 76 on any card by using "%%". (In some of the pseudo order types, only one % is required to terminate it.) Comments may be inserted after

such a termination. The % after the last formula (or statement, etc.) on a card may be omitted.

The identification field (cols. 77 - 80) is never used as part of a program. Anything desired may be punched into these four columns. To simplify the key punching of FORAST programs, it is recommended that these four columns be used only for a decimal numbering of the program cards. (This numbering may be reproduced rather than key punched on the cards and need not be written by the programmer.) Error prints obtained during translation of a problem will also print the identification field of the card that contained the error. (See section XI).

VIII. USE OF LOCATION FIELD

The location field (cols. 1 - 6) may be used to give a symbolic name to the first instruction or the first number that is coded from a card or it may be used to specify an absolute storage address for the orders and/or numbers that appear on the card and on the following cards.

The location field is ignored when it is blank. It is also ignored when it has the same name (symbolic or absolute) as the last preceding non-blank location field. This allows extra cards to be inserted in front of a card that has a location name and the location name designates the first of the cards that have the same location name; this facilitates insertion of a temporary PRINT statement for checking. If some other location name is used between the two locations that have the same name, then the code generated at the second location will be stored over the code generated at the first location and will destroy it.

The location field controls an absolute machine address which shall be referred to as the "location counter." This address normally

starts at 0100 (sexadecimal, 01040 for BRLESC) and is advanced by one for each machine word that is generated by the FORAST translator from the FORAST program. If any decimal or sexadecimal absolute address appears in the location field, then the location counter is set to that address. If this is done, the old location counter address is not remembered by the translator and all following generated code will be stored consecutively from the new address until a location field is encountered that will cause the storing to begin elsewhere. Thus changing the location counter may control the storage of many following cards, not just the card on which the new location appears. If a symbolic address that has not yet been assigned appears in a location field, it immediately becomes assigned to the address that is in the location counter at that time. Thus the assignment of machine addresses to all names that appear in the location field is done as soon as these names are encountered. If a symbolic address that has previously been assigned (by being a previous location or in a SYN or BLOC statement) appears in a location field, then the location counter is set to the address that was previously assigned to this symbolic name.

Special rules apply to names in location fields that are within a previously defined "BLOC". (See BLOC page 44.) If an unassigned block address is used in a location field, then the initial name of the block is assigned to the current value of the location counter and the location counter is then advanced to the actual address within the block that was used in the location field. For example, if X1 - X4 was defined as an unassigned block and the location counter was currently at 0142 and then if X3 was used as a location, X1 would be assigned 0142 and the location counter would be advanced to 0144, which is the address of X3, and the next generated code or number would be stored in X3. Thus space is allocated for a block up to the block name used but not beyond it. If

space should be left for the entire block, then the name of the last cell in the block must be used in the location field. If a location block address has previously been assigned, the location counter is set to the actual assigned address within the block.

Increments may be used on symbolic location addresses. If the symbolic address has been assigned, then the location counter is set to that address plus or minus the increment. Thus A + 2 would set the location counter to 0202 if A was previously assigned to 0200. If the symbolic address has not been assigned, then it is assigned to the location counter first and then the increment is added to or subtracted from the location counter. Thus a positive increment on an unassigned address causes the location counter to skip ahead and a negative increment causes it to be set back and probably causes some previously generated code to be destroyed.

SELF is a symbolic name that may be used to refer to the location counter. It cannot be used for any other purpose. If SELF is used in any instruction or statement, it is temporarily assigned to the current location counter address. (In assembly orders, it is the location of the order that it is used in.)

Since ORDVAC is a single address computer with two orders per word, it is necessary to have some special location field rules so that the programmer will have some control over the storage of left and right orders. Since BRLESC is a three address computer with one order per word, the special rules in this paragraph do not apply to BRLESC. If a location field on ORDVAC contains either an absolute machine address or an unassigned symbolic address that does not begin with the letter R or a decimal digit, then the next order coded will be a left order. If an

unassigned symbolic location address begins with R, then the next order coded will be on the right side. If the symbolic location address begins with a decimal digit, the next order will follow the previous order and hence may be either left or right. Thus orders that should be coded on the left side of a word should be given a location that begins with a letter other than R and orders that should be coded on the right side should have locations that begin with R. If the next order may be coded on either side, then it is best (but not necessary) to use a location name that begins with a decimal digit. The ORDVAC FORAST translator codes a ZX (SELF + 1) conditional stop order whenever it inserts a dummy order so that the next order will be on the proper side. These special location field rules apply only when ORDVAC is generating orders, not when constants and full words are being stored by DEC, SEXA, etc. pseudo order types. These full word constants always occupy a full word and the location counter will be advanced by a half word if necessary before storing a constant. The ORDVAC dictionary listing prints L's and R's to indicate left and right location names. SELF is used as a left location regardless of which side the location counter is currently on.

On BRLESC, symbolic index names must not appear in a location field unless they have been previously assigned or used as an index register. (Index register names get assigned on BRLESC as soon as they are used as an index register.) If an assigned index register name is used in the location field, the location counter is set to the assigned address.

It is not necessary for all symbolic addresses to appear in a location field. The translators automatically assign storage space for all symbolic names that remain unassigned when the END of the program is reached. This assignment of all of the rest of the unassigned names begins with the address that is in the location counter after the END card (See END page 68) is processed, hence the location counter must be left at some address that has enough space after it for assignment to all of the un-

assigned names. This automatic assignment of addresses is done so that all names have unique storage except for those names that appear within SYN statements (See SYN page 48). Enough storage space is always left for all blocks and when SYN is used to make a name in a smaller block (or a non-block name) the same as one in a larger block, the larger block is assigned first so that the smaller block will fall within the larger block. If neither of such blocks is completely contained within the other, enough space is left to provide storage for all of both blocks. This machine assigning is done in the sequence the names appear in the dictionary except for the names that appear in SYN or LAST pseudo order types. (BRLESC will assign some and possibly all of the single variable names between the constant pool and the subroutines). Hence if certain names or blocks must be assigned in a definite sequence in the memory, the programmer should use these names in location fields or on a "LOC" card (See LOC page 49) to insure that they are assigned to the proper sequence of memory positions.

IX. Pseudo Order Types

The order type field (cols. 7 - 10) may be used for any of the pseudo order types that are defined below. There are two major types of pseudo order types; (1) there are those that do nothing but allow the programmer to control to some extent the translation of his program and (2) there are those that allow constant information to be stored as part of a FORAST program. On every card, the order type determines the type of information the translator expects to find in the formula and statement field (cols. 11 - 76). The list of permitted pseudo order types is: (1): PROB: BLOC: SYN: LOC: LAST: CONT: LIST: END: DATE: COMM: MODE: STOR: NOS.: FTTS: ASGN: SUBR: O.T. (2): DEC: DEC=: FORM: SEXA: ALFN

A. PROB

A PROBlem card should be put at the beginning of every FORAST program to identify the program. It should contain the problem number, the programmer's name (or at least initials), the approximate date it was programmed and a brief title or description of the problem.

This information is not used by the ORDVAC FORAST translator except that it is printed out ahead of the dictionary and/or the problem output to identify these outputs.

For BRLESC, a PROB identification card is mandatory and a proper problem number must be recorded on the PROB card after "PROB". A program that does not have a PROB card before the first formula (or the END card) will not be compiled or run.

The problem number, to which the computer time is to be charged, should be the first thing after column 10 and must not extend beyond column 20. If other characters follow it before column 21, there must be one of the following characters at the end of the problem number: blank - + ( ) % or comma. If any of these characters are inserted before or between the first three characters of the problem number, they will be ignored.

If more than one PROB card is used in one program, the first one is the one that will actually be used. The others will be ignored. (Any PROB cards that have cols. 11-20 blank will be ignored.)

The reason for mandatory PROB cards is that the BRLESC FORAST compiler and N. PROB subroutine make use of the real time clock to keep a record of the computer time that is required to run each problem. This record consists of punching the PROB card at the beginning of the problem with cols. 61-70 replaced with the date and cols. 71-80 replaced with the "start time". At the end of the problem, another card is punched that contains the problem number in cols. 1-6, the "charge time" in cols. 7-10 as hrs. and mins., the total time, the compile time, the date, and the "stop time" in cols. 71-80. These two cards for each problem will be punched into a special hopper on the card punch unit and thus will not appear with the normal outputs.

When the C. PROB subroutine is used to compile several programs consecutively, the BRLESC time will normally be charged to the problem number that is on the PROB card in the last program. However, all of

the PROB cards should have the same problem number and the PROB card of the first program is the only one that will be punched for the time-keeping record with the start time on it. The compile time will be only for the first program compiled.

If you have a legitimate reason for not being charged for running your problem, a card having "NO CHARGE" in the cols. 11-76 field may be inserted to cause the BRLESC charge time to be zero.

The location field of a PROB card is always used. A PROB card that is blank in columns 11-20 is ignored except the location field is still used. Example: PROB 647.1 J.Q. BROWN JULY 1961 AIR FLOW

B. BLOC

This is used to define the names and sizes of one or two dimensional blocks of storage. Two dimensional blocks of storage will be referred to as arrays.

One dimensional (linear) blocks are defined by writing the symbolic names of the block followed immediately by the initial decimal integer "subscript". (The word subscript will be used here to refer to the decimal digits, however the subscript is written on the same line as all of the other characters in the name.) Thus A1 could be the name of the initial cell of a block. A dash (minus sign) is used to separate the initial block address from the final block address. The final block address must have the same letters as the initial address but they are followed by the final decimal subscript. Thus A1 - A10 would be the definition of a linear block of ten memory positions and each position in the block may be referred to in the rest of the program by using the names A1; A2; A3; A4; .... A10. Note that A and A12 are not names that are a part of this block and may be assigned memory positions that are quite different than those assigned to the block A1-A10. The initial subscript may be blank or zero or any positive decimal integer and the final subscript would normally be larger than the initial subscript.

(Only BRLESC allows the final subscript to be smaller in which case the smaller subscripts are assigned to larger addresses than the larger subscripts.) If the initial subscript is blank (omitted), it is used as zero but has the additional effect of allowing the initial cell of the block to be referenced by no subscript or a zero subscript. Hence if B - B6 is defined as a block, the initial cell may be called either B or B0.

The complete block name, including the largest subscript, must not be more than six characters on ORDVAC. On BRLESC, a total of eight characters is allowed with the following restrictions; if the block name is three or less characters, the subscript may be any 5 digits, for four letter block names, the subscript must not be larger than 4095 and for five letter names, the subscript must not be larger than 63.

The names within a linear block will be assigned to consecutive memory positions unless the block definition is followed by a / symbol and a decimal, or sexadecimal or a previously assigned symbolic name (that may have an increment) that determines the spacing between each element in the linear block. Hence T0 - T20/2 may be used to specify a linear block of 21 memory positions that uses every other position, i.e. if T0 is 0200, then T1 is 0202, T2 is 0204, etc. If a symbolic name is used to indicate the spacing, its previously assigned address (not its contents at run time) is used, i.e. the bloc spacing is fixed at compile time. (A SYN statement would normally be used to assign a symbolic name for this purpose.) Non-consecutive spacing is allowed on linear blocks so that several of them may be "interwoven" by using a LOC pseudo order type. (See LOC page 49).

A linear block definition may be preceded by "absolute address /" or "I/". The absolute address will be assigned to the initial name of the block and the "I" will cause the block to be assigned to index registers. (The "I/" is only necessary when the block must be assigned to index register memory. Thus the full general form of a block definition is:

(I or mach. add./initial name - final name/spacing)

Some examples of linear block definitions would be:

BLOC(B1-B22)(A-A420)MAT 5-MAT 20)T1-T60/3)
BLOC(I/I1-I4)0600/3R-3R199)

Two dimensional blocks (arrays) may be defined and referenced by writing a symbolic name and a row subscript followed by a comma and a column subscript. M1,1 - M4,4 would define an array that has four rows and four columns and requires sixteen consecutive memory positions. All arrays must use consecutive memory positions and are stored by rows, i.e. the names of consecutive positions of M1, 1-M4,4 would be M1,1: M1,2: M1,3: M1,4: M2,1: etc. The initial and final row subscripts can be zero or any positive decimal integer. (The initial column subscript must be less than 64 and the final column subscript must be less than 256 plus the initial column subscript.) Arrays may be square or rectangular and may also be defined as triangular by following an array definition with "/SY." (symmetric) or "/LSY." (lower symmetric). The upper triangle is stored when SY. is used and the lower triangle is stored when LSY is used. For SY. arrays, the column subscript must be greater or equal to the row subscript and for LSY. arrays, the row subscript must be greater or equal to the column subscript. SY. arrays may have more columns than rows (may be augmented) but LSY. arrays cannot have more columns than rows.

The symbolic letter positions of array names should not have more than four letters (small arrays of less than 64 memory positions may generally have five letters) and the letters should be different than the letters used for any linear blocks. (FORAST translators handle array addresses by "linearizing" them and linear block names must therefore be different than any array address that has been "linearized." Thus M15 is the same as M4,3 in the array M1,1-M4,4 and must not be a part of any

linear block.) Arrays cannot be assigned to index register storage. Some examples of array definitions are:

```
BLOC(R1,1-R4,6)(0800/AT1,1-AT10,5)
BLOC(BQ1,1-BQ10,11/SY.)MAT-MAT5,5/LSY.)
```

Array addresses may be indexed by using a second comma after the array name followed by an index register name. Thus R1,1,I illustrates the method of indexing R1,1 by I. If the index register address used to index an array address is decimal or sexadecimal, it must be enclosed in parentheses. Hence R1,1,(10) is the way R1,1 can be indexed by index register ten. (Note that R1,1 would be R1 indexed by index register one if R1,1 was not defined as part of an array.) Indexing in FORAST should not be thought of as variable subscripts, it is simply the addition of a variable integer to a primary address that determines the actual address used at run time. This means that A1 is not necessarily the same as A,I when I contains a one, they are the same only if the initial cell of the block is called A. If the initial cell is called A1 and it is desired to reference A1,A2, etc., then it should be written A1,I where I assumes consecutive integer values starting at zero. This emphasizes the fact that the subscripts used with the letters of a block or array name must "fall within" the block or array before the name is a member of the block or array.

A block or array definition (BLOC card) must precede any reference to members of that block or array. It is wise to define all blocks and arrays before writing any other symbolic addresses.

Columns 11 - 76 of a BLOC card may be used to define one or more blocks or arrays. Successive definitions should be separated by a right parenthesis (a left one is optional) and a "%" may be used after the last one to ignore the rest of the card. The location field should not be used.

C. SYN

This may be used to assign absolute addresses to symbolic names or to allow different symbolic names to be assigned to the same memory space. For some problems, the memory may not be large enough to allow a unique position for each and every number, thus it may be necessary (to avoid using drums or tapes) to use the same memory position for more than one number when such numbers are not needed at the same time. A and Q normally would be assigned to two different memory cells but if A is computed and completely used before Q is computed, then it would be all right to store Q in the cell that previously contained A. A SYN statement of (A=Q) would cause the symbolic names A and Q to be assigned to the same memory cell.

Each synonym definition is of the form:

(Add. = Add. = ... = Add.)

where Add. may be any type of address allowed (absolute or symbolic) but <u>increments cannot be used</u> in a SYN statement. Addresses within blocks or arrays may be used. The same effect as increments can be achieved by defining one or more "false blocks" that are then used in SYN statements to obtain the desired storage arrangement. However the LOC pseudo order type is usually sufficient for this purpose. (See LOC page 49).

SYN cannot be used to reassign any address that has been previously assigned. (See ASGN page 73). There must not be more than one address in each synonym definition that is an absolute address or has been previously assigned. As soon as any one name within one synonym definition becomes assigned, then the other names are assigned accordingly.

The addresses in SYN definitions are always assigned properly (no unexpected overlapping of storage) if the machine assigns the first address in that definition after the END card. However, if the programmer causes one address in a definition to be assigned by using it in a location field (or another SYN statement, etc.), then the other addresses are assigned accordingly without any checks for overlapping storage. When the machine assigns a name that is involved in a synonym definition, it allows enough space for all of the blocks or arrays that are assigned because of the synonym definition and smaller blocks or single names are always assigned within the larger blocks.

Each synonym definition should be separated from the next one by a right parenthesis though a leading left parenthesis is optional. A "%" after a definition causes the rest of the card to be ignored.

Some examples of some SYN definitions are:

```
SYN(A=B=C)R=20)K1,4=DELTAV %
SYN(G14=E10=F1=T)(T=T1=Q)
```

The location field of a SYN card should not be used.

The ORDVAC FORAST translator has a limit of 64 (55 if the computed GOTO statement is used) unassigned symbolic names that are used in SYN definitions. Thus an ORDVAC program must not use more than 64 (or 55) names in SYN definitions until some of them are assigned. BRLESC allows 288 unassigned SYN names.

D. LOC

A LOC card allows many "locations" to be specified on one card. The location field is first processed in the normal way and then the addresses in cols. 11-76 are also processed as "location fields". (See VIII page 38).

A LOC card allows a programmer to cause a list of symbolic names to be assigned in a desired sequence. Successive names will normally be assigned to consecutive memory positions although block or array names, increments, and previously assigned addresses may cause non-consecutive assignment. Note that space for the symbolic names is allowed where the LOC card appears in the program and the location counter is advanced by one beyond the last space assigned on that LOC card.

One of the primary uses of a LOC card is to define the desired sequence of "interwoven blocks". To cause (X-X20/3)(Y-Y20/3) and (Z-Z20/3) to be interwoven blocks (a spacing of 3 must be specified in a BLOC statement), (X)(Y)(Z20) must be specified in a LOC statement. Note that the first name of a block is used for all blocks in an interwoven string of blocks except the name of the last cell in the last block is used. The example above causes X to be assigned to the location counter, the location counter is advanced by one and Y is assigned. The location counter is again advanced by one, as it is after every address on a LOC card, and the Z block is assigned beginning at the current address in the location counter. Then since Z20 was written instead of Z, the location counter is advanced to Z20 and again advanced by one before assigning or storing anything else. Hence the use of Z20 allows space for all the Z's which includes space for all the other X's and Y's. The sequence of these block names in the memory would be X,Y,Z,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3,....,X20,Y20,Z20.

Only one address may be written in the normal location field. Each address in cols. 11 - 76 must be followed by a right parenthesis with an optional leading left parenthesis. A % after the last one will cause the rest of the card to be ignored. Increments are allowed. A blank

location field, or more than one right parenthesis between names causes the location counter to be advanced by one. Thus (A)))B would cause two blank locations to be left between A and B.

Some examples of LOC cards are:

X LOC (Y) Z20) R1) S1) T300) Q) TL $

P LOC (V1)(V2)3AA) $

X10 LOC(Y10)Z10 $

This last example causes the blocks (X1-X10)(Y1-Y10) (Z1-Z10) to be stored consecutively providing none has been previously assigned. The same effect using the "false block" technique is achieved by defining a "false" BLOC(F1-F30) then SYN(F1=X1)(F11=Y1)(F21=Z1)

E. LAST

This pseudo order type may be used to tell the translator that a certain block (or several blocks that are interwoven) or one array should be assigned after all other storage has been assigned. It allows a programmer to reserve the "rest of the memory" for some data that is of indefinite length. Thus a programmer may allow as much space as possible for a specific purpose without knowing ahead of time just how much space is available.

The name or names that should be assigned last are to be written in cols. 11 - 76 and separated by a right parenthesis. There can be only one array name or there can be several block names if they have non-consecutive spacing (interweaving) in the BLOC definition. There can also be a list of names or just one name that is indexed in the running program to access or store the rest of the data. If a block or array name is used, any name within the block or array may be used.

If more than one block name is used, the first elements of each block names are assigned to consecutive positions regardless of the subscript used. (It is not necessary to use the last subscript of the last block named.) Since a LAST card essentially assigns a "block of storage", the name(s) need not also be defined on a BLOC card unless the program explicitly uses the names of elements in the block other than the name of the first element. BLOC definitions must always have a numerical maximum subscript, but for those block names that are used on a LAST card, the BLOC definition need only include the largest subscript actually used anywhere in the program.

If several names are written on a LAST card, they are assigned to consecutive positions in the same sequence they are written. If other names have been defined as synonyms (SYN) of LAST names, then they are not assigned until the LAST names are assigned.

There is a restriction of a maximun of 63 names on a LAST card. (BRLESC allows 111 names.)

The location field should not be used and a % after the last name causes the rest of the card to be ignored.

Some examples of LAST cards are:

```
LAST A1,1 %
LAST (X1)Y1)Z1)
```

F. CONT

This pseudo order type should be used when it is necessary to continue from one card onto the next card. Columns 11 - 76 of a CONT card continue from column 76 of the preceding card. The translator assumes a % character after column 76 of each card except when the next card is a CONT card. It may be used after any card that uses columns 11 - 76 except that it cannot be used to continue a "string of characters" in a PRINT or PUNCH statement.

It must be used whenever a statement or formula is started on one card and completed on the next card. A CONT card is ignored if column 76 of the previous card was ignored, thus comments may be continued also.

The location field should not be used. There is no limit to the number of successive CONT cards allowed.

G. LIST

A LIST card may be used to tell the translator to print a "dictionary" that shows the assignment of all symbolic addresses for the program that is being translated. The dictionary is not printed unless a LIST card is inserted somewhere in the program (or the translator finds an error in the program).

A sexadecimal listing of the code may also be obtained by writing "S. CODE" in columns 11-76 and a binary card program deck may be obtained by writing "B. CODE" in columns 11-76. On BRLESC, if a program is written for use as a subroutine (with key words, relative addresses, etc.), "B.SUBR" may be used to obtain a binary deck suitable for use as a standard relocatable subroutine. "NO DICT" is also allowed in case a dictionary is not desired when obtaining a sexadecimal or binary print of the program. All other names on LIST cards are ignored. If more than one type of listing is desired, the names may be on the same LIST card separated by a right parenthesis or they may be on separate LIST cards. A "B.CODE" print is not done if a "B.SUBR" print is done for the same program.

Each line of an "S.CODE" listing from BRLESC consists of a location address followed by four words that are stored beginning at that address. It is a "memory dump" type of print that shows the program as it is stored in memory. It includes index registers 008-03F

and from 080 to the beginning of the subroutines. Any line that would only show four zero words is not printed.

A B.CODE print is a memory dump onto binary cards and it includes the subroutines used by a program except for the decimal input-output routine. To use this deck to run a program, a program input routine must be placed in front of it and the decimal in-out routine must be placed behind it. The last card of this program deck must be a key word that jumps to 073. Since the BRLESC translator is extremely fast, the use of binary decks is unnecessary. (For some problems, it will take longer to read the binary deck than it would take to read and translate the symbolic deck.) A B.CODE print includes 00K-03F; 058-06L; 07J-07L; 0S0-S9L and 01040-end of memory except for the next to last group of 4096 cells. (Zero words are not punched.)

On BRLESC, a B.CODE or B.SUBR print is done between the dictionary print and the S.CODE printing. The binary cards will always have a y punch in column 80 and will be separated from the other cards by a blank card.

On the ORDVAC, sexadecimal print is a memory dump of 040 to the beginning of the subroutines and prints eight words per card with a "range card" at the beginning of each group of cards. The range card specifies the first and last locations of the following group and no group exceeds 256 words. Cards that would show only eight zeros are not printed when there is more than one such card in succession. These sexadecimal cards should be listed on the 1401 to get extra spacing.

The ORDVAC binary print is a memory dump of 000-009; 040-J7L and OLN1-OLNJ. To use this deck, it must be preceded by a program input routine (5 key Input Routine) and the decimal input-output routine must be inserted in front of the last binary card. (The next to last card of the binary deck ends with a OJJO store key word and the last card ends with a key word that jumps to 009.)

When the dictionary is listed on either computer, four symbolic addresses are listed on each line and the dictionary is in alphabetical order by columns. (The numerical value of the six bits used to represent each character determines the alphabetical order, hence the characters zero and prime are between R and S and a decimal point is between I and J.) Each symbolic name is followed by its sexadecimal assigned address and this may be followed by any of the following letters:

L - Location name. (Left location on ORDVAC)

R - Right location on ORDVAC.

B - Block name. (Both initial and end names appear in the dictionary, the assigned address printed with the end address is the spacing within the block).

A - Array name. (The linear equivalent of both the initial and end names of an array appear in the dictionary. The assigned address printed with the end address contains the length of a row as the last two sexadecimals and is preceded by the initial column subscript.)

S - Appeared in a SYN statement.

M - Machine assigned address.

I - Index Register name. (BRLESC only)

F - Function or Subroutine name.

U - Unused name. (This means that the name appeared only once in the program and may indicate a programming or key punching error. It does not keep the program from being run.) A dictionary may be listed on the hi-speed printer or on the 1401.

The dictionary will include two extra entries at its end. One is %NOS. with an address that indicates the end of the constant pool. The other one is %SUBS. with an address that is the beginning of the subroutines. BRLESC will also print %INDEX with an address that is one more than the last index register used by the program.

The location field of a LIST card should not be used. Some examples of LIST cards are:

```
LIST
LIST(B.CODE)(S.CODE)%
```

H. DEC

When this pseudo order type is used, columns 11-76 may be used to write one or more decimal number that are to be converted to binary numbers and stored in consecutive memory positions beginning at the address in the location counter. Thus the location field may be used to specify the address or name of the first number on a DEC card. Each number should be separated from the following one by a right parenthesis, a leading left parenthesis is optional. A % after a number causes the rest of the card to be ignored. Extra right or left parentheses do not cause memory space to be skipped. Commas and leading zeros may be used on any part of any number.

FORAST allows three types of numbers; (1) floating point, (2) fixed fraction, and (3) integer numbers. Any type of number may be written anywhere on a DEC card.

1. Floating point numbers have the general form of

$$F \pm dd \ldots d \pm e \ldots e$$

where $\pm$ dd..d is the coefficient and $\pm$ e ... e is a power of ten

exponent and may be omitted if it is zero. The leading F is not necessary if the MODE arithmetic (See MODE page 69) is floating point. The leading + sign on the coefficient is optional. A decimal point may be punched anywhere in the coefficient, but if none is punched, it will be assumed to be at the right end of the coefficient.

If the MODE arithmetic is floating point then all of the following examples are floating point numbers:

DEC(10)F-4.1).92-04)2,462,147)-1.+5 %

On BRLESC only, an exponent may be started with an E and floating point numbers may be followed by a U and a positive integer that indicates the power of ten by which the number should be unnormalized.

2. Fixed point fractions may have the general form of

X ± dd ···· d ± e··e B ± s··s D ± r··r

where ± e··e is a power of ten exponent, ± s··s is a binary scale factor and ± r··r is a decimal scale factor. The exponent, both scale factors and leading + signs (except on the exponent, to show where the exponent begins) are optional. The leading X is optional if a B or D scale factor is specified or if the MODE is fixed point arithmetic. ORDVAC fixed point fraction numbers must be less than one in absolute value after the scale factors are applied. BRLESC fixed point fraction numbers may be as large as sixteen in absolute value. If the MODE is fixed point, fixed point fractions must have either a decimal point in the coefficient or a scale factor. If the coefficient does not contain a decimal point, it will be assumed at the right.

Examples of fixed point decimal fractions are:

DEC(X4.2B-5)-31.7B-3D-2)+.17-2B+2) %

3. Integer numbers may have the general form of

$$I \pm dd \cdots d$$

where the + sign is optional. A leading I must be used when the MODE is floating point arithmetic. A decimal point is ignored when a leading I is used. A decimal point is not allowed if the I is omitted and the MODE is fixed point arithmetic. Integers are scaled at 2^{-39} on ORDVAC and 2^{-60} on BRLESC. ("Integers" with other scaling can be written as fixed point fractional numbers.)

Examples of integer numbers are:

DEC(I10)(I-147)I+41)I24,861 %

I. DEC=

This pseudo order type is just like DEC explained above except it allows each number to be preceded by a location name and "=". Thus the general form of each number is:

Loc. name = any DEC number

DEC = allows each number to be given a name without generating any running code or using a separate DEC card for each number. Each number must be preceded by a "name =", or just "=" if no name is to be given to a number. Each number is separated from the next one by a right parenthesis with a leading left parenthesis optional. A % causes the rest of the card to be ignored.

Some examples of DEC = cards are:

DEC = (X=4.7)(J3=I13)EPS=.3-6)TX=X.51B-4 %
DEC = (A1,1=1)(A1,2=17.4)A2,1=-6)A2,2=+9.1 %

The first characters on either a DEC or DEC = card in columns 11 - 76 may be either ORDVAC or BRLESC. If one of these names is used, then the numbers on the card are stored only on the machine whose name

was used. Thus it is possible to write one program that uses different constants on different machines. This is sometimes desirable because of the difference in word length (68 bits vs. 40), difference in floating point number range ($10^{\pm 155}$ vs. $10^{\pm 38}$), difference in speed (20 to 1), etc. between BRLESC and ORDVAC.

J. FORM

This pseudo order type may be used to specify FORMAT words that may be needed for READ and PRINT or PUNCH statements. The FORMAT words are used to describe the type of each field, (T), the length of each field (L), and sometimes a scale factor (S). The location field may be used to give a name to the format being specified. Successive field definitions may be written in columns 11 - 76 with each one separated by a right parenthesis with an optional leading left parenthesis. A "%" causes the rest of the card to be ignored.

Each field definition may have one of three forms:

(T)	just a type
(T-L)	type and length
(T-S-L)	type, scale factor and length.

T,S, and L may be specified only by decimal or sexadecimal numbers. (Sexadecimal numbers must have a leading zero.) Numbers within a field definition are separated by a dash (minus sign).

Example: FORM (9-12)1-2)(6-3-10)2 %

The types of fields allowed are as follows:

T = 1 Repeat the previous fields, beginning with the field after the last repeat type, L times where L <256. If there are no previous repeat types in the format, it is repeated from the beginning.

T = 2 This type marks the end of a format definition. If more numbers remain to be read or printed, the format is repeated from the beginning and a new card is begun. If any FORM card does not have a 2 type as its last type, the translator automatically adds a 2 type unless the next card is a CONT card. Thus if a single format definition requires more than one card, all cards after the first one should be CONT cards. S and L are not used.

T = 3 Skip L columns where $L < 256$.

T = 4 Integer field.

Input: Read an integer that is punched anywhere (may include the sign column) in L columns and store the number as an integer ($L < 256$). Blank columns before and after the integer are ignored. Digits to the right of a decimal point or to the right of a blank column in the middle of the number are ignored. The sign may be punched in any column of the field that is above or to the left of the leading digit.

Output: Take an integer number from the memory and print it at the right end of L columns. The sign will be punched in the leftmost column and will not be double-punched (a sign and digit in the same column) unless the double punch option is being used. Columns to the left of the integer will be blank.

T = 5 Print card counter.

Input: Skips L columns. (L < 16)

Output: Increase the absolute value of the card counter by S and print it in L columns. The card counter is kept as a fixed point fractional number (at 067 on BRLESC and OJJ2 on ORDVAC) scaled at 10^{-L}. The length of the counter field should not be changed unless the counter is set to zero. (The ZEROCC subroutine should be used to set it to zero.) If the output does not use the whole card, the counter is printed at the right end of the card regardless of where the format might say it should be printed. It will not be printed if there are not enough columns left to print it. The counter may be negative and leading zeros are printed.

T = 6 Fixed point fraction.

Input: Read a fixed point number from L columns, assume a decimal point after S columns from the left, (not counting the sign column if single punched signs are used) and store a fixed point fraction number. If a decimal point is punched in the field, it is used instead of the assumed point after S columns.

Output: Take a fixed point fraction from the memory and print it in L columns. S is ignored. Leading zeros are printed.

T = 7 Fixed point fraction with decimal point.

Output only: Take a fixed point fraction from

the memory and print it in L columns with a decimal point printed after S columns. (The number printed will always be on the right of the decimal point except on BRLESC where the number can be as large as sixteen.) Leading zeros to the left of the decimal point are printed.

T = 8 Alphanumeric

Input: Read and store the six-bit representation of the characters punched in L columns. Only the rightmost 60 bits of BRLESC words are used to store alphanumeric characters. If L < 10, the characters will be in the left portion of the sixty bits. If L > 10, the characters will require more than one computer word and will use as many consecutive memory cells as are required. Each new field starts exactly at the address specified in the entrance sequence regardless of the length of the previous field and begins storing in the left part of that word.

For ORDVAC, the rules are the same except the rightmost 30 bits of ORDVAC words are used and hence a maximum of 5 characters are stored in each word. A new consecutive word is used after each group of five characters are stored and the number of characters stored is not necessarily a multiple of ten.

Output: Take L alphanumeric characters from the memory and print then in L columns. Characters are taken from left to right from the rightmost 60 bits of BRLESC words. If L > 10, characters are taken from the next consecutive word or words until L characters have been printed.

For ORDVAC, the characters are taken from the rightmost 30 bits and a new consecutive word is used after each group of five characters.

Note that for both input and output, only one address is used from the list in the READ, PRINT or PUNCH statement for each alphanumeric field regardless of its length.

T = 9 Floating point number with exponent.

Input: Read and store a floating point number from L columns where the last 3 columns (2 columns if use double punched signs) contain an exponent. A decimal point is assumed after S columns. If any column actually has a decimal point punched, it is used and S is ignored.

Output: Take a floating point number from the memory and print a floating point number in L columns with an exponent in the rightmost 3 columns (2 columns if use double punched signs). The exponent is decreased by S before printing but a decimal point is not printed after S columns. Leading zeros of positive exponents are not printed and the coefficient part of the number never has leading zeros unless the number is zero. A zero number will print a coefficient of all zero characters.

The standard FORAST output format defines 6 of these 9 type fields of 12 columns each.

T = 10 or OK Input: Read a number from L columns and store it as a floating point number. The number may be punched in either fixed point form or floating point form with an exponent. Blank columns are

ignored and either coefficient or exponent may start in any column within the field. If there is an exponent, it must begin after the first blank column or punched sign that occurs after the coefficient. A decimal point is assumed after S decimal digits that are punched for the coefficient or if a decimal point has been punched on the card, it is used and S is ignored. Signs may be either single punched or double punched regardless of how this option is set except that a zero must not be punched under a sign (BRLESC restriction). (The zero would be used on ORDVAC and ignored on BRLESC.)

This type of field is used as the standard FORAST input field. The standard input format defines 6 of these 10(or OK) type fields of 12 columns each.

Output: Take a floating point number from the memory and print it as a floating point number with an exponent in the last 3 (2 if use double punch signs) columns and a decimal point printed after S columns of the coefficient.

T = 11 or 0S

Input: Read a fixed point number from L columns and store it as a floating point number. A decimal point is assumed after S columns of the number. (Don't count a single punched sign column). If a decimal point is punched on the card, it is used and S is ignored.

Output: Take a floating point number from the memory and print it as a fixed point number in

L columns. The number is aligned, if possible, so that its decimal point is after S columns of the number but the decimal point is not printed. If the number has more than S digits before the decimal point, the decimal point will actually be printed where it belongs. On BRLESC, if the decimal point falls outside of the allotted L columns, the number is printed as a floating point number with an exponent (if there are enough columns). Leading zeros are punched.

T = 12 or ON

Input: Not allowed on ORDVAC input. On BRLESC, it is the same as type 10(OK) except when nc decimal point is punched in the field, it is assumed to be at the right end of the number. (S is ignored)

Output: This is the same as type 11(OS) described above except that the decimal point is actually printed and leading zeros are not punched or printed.

T = 14 or OF

Input and Output: This indicates "end of card (or line)". A new card (or line) is started without starting at the beginning of the format. This allows multi-line formats to be written without the necessity of using all of the columns on each line. If this type 14 is used at a point where a new line would begin anyway, then it is ignored. This means that a type 14 at the beginning of a format is useless and successive type 14's do not cause any blank lines. At least one column must be used or skipped between 14's to get a blank line (or read an extra card.)

When a type 14 is used in conjunction with a subgroup specification, the subgroup count does not start over at a type 14; it continues counting and causes a new line and the re-use of the beginning of the format when it is exhausted

T = 15 or OL

Input: Not allowed.

Output: Allowed only on BRLESC when the F.T.PR (firing tables print) subroutine is being used by the insertion of a SYN(F.T.PR = F.T.PR) statement in the program.

This format type can be used to change the sign option or the double punch option anywhere on a card. S is a six bit character (written in decimal or sexadecimal) that will be used for the plus sign and L is a six bit character (written in decimal or sexadecimal) that will be used for the minus sign. (A y row sign is 010 or 16 and an x row sign is 020 or 32.) A fourth number of 0 or 1 may be specified after the L to indicate the setting of double punched or single punched signs respectively. All of these options apply only for the current PRINT or PUNCH statement, a new statement returns them to their previous values.

F.T.PR Modifications:

When the F.T.PR subroutine is used on BRLESC, (by insertion of a SYN(F.T.PR = F.T.PR) statement in the program) the above rules and formats are modified in the following ways:

1. Adding 040 or 64 to any of the format types that control printing of numbers will cause a sign that is the opposite of the actual sign of the number to be printed in an extra column at the right end of the number.

This extra column will not be included in the number of columns specified by L. The normal sign will precede the number and if the number is zero, neither sign is printed.

2. Adding 080 or 128 to any of the format types that control printing of numbers will cause the leading sign to print immediately to the left of the first digit of the number; i.e. it prints a "floating sign".

3. All numbers will have at least one digit printed in front of the decimal point. For numbers less than one, a zero will be printed before the decimal point unless there isn't space for it.

4. A number printed as zero will not have any sign printed.

5. A floating point number greater than 10^{51} will cause a blank field to be printed in its place.

6. All numbers will have leading zeros printed as blanks up to the decimal point or the first non-zero digit, with the exception of one zero digit immediately before the point.

K. SEXA

This pseudo order type may be used to store sexadecimal constants. Columns 11-76 may be used to write one or more sexadecimal constants with each one enclosed in parentheses. (The leading left parenthesis on each one is optional.) A % causes the rest of the card to be ignored. The first symbols appearing in columns 11-76 may be either "ORDVAC" or "BRLESC" and will cause the card to be processed only on the computer whose name was used. Since ORDVAC words are ten sexadecimal characters long and BRLESC words are seventeen sexadecimal characters long, it is usually necessary to write separate cards for each computer.

The characters 0 to 9 and K,S,N,J,F,L are used to represent the sixteen sexadecimal digits. If a constant is written with fewer characters than required to make up one computer word, the digits will be placed at the right end of the word, thus (3) is the same as (003) and is also the same as an integer number 3. The character Z is also allowed but may be used only once in a constant. The Z represents a string of zeros that is long enough to fill out the computer word. Thus (6Z8) will have a leading 6 followed by zeros until the last digit which will be an 8. A Z may be used to replace one string of zeros in a sexadecimal constant.

The location field is used as the location of the first constant on the card.

Some examples of sexadecimal constants are:

SEXA(52K)921J0F4L)4)
SEXA BRLESC(032 Z4K)KKZ)4LL)%

On BRLESC only, an A may be used to represent five sexadecimal zeros and an M may be used to represent five sexadecimal L's (fifteens) in sexadecimal constants.

L. END

The last card of every program must be an END card. It causes the translator to stop reading symbolic program cards, to complete the translation of the program that has just been read and to then start running the program. Cols. 11 - 76 may contain a GOTO statement that specifies the location of the first statement or instruction that is to be done in the running program. If no GOTO appears, a GOTO (0100) is done on ORDVAC and a GOTO(01040) is done on BRLESC. Note that the first statement to be done does not have to be the first physical statement in the program. A % causes the rest of the card to be ignored. BRLESC allows GOTO(N.PROB) or GOTO(C.PROB) to be used on the END card.

The location field of an END card is used in the normal way. If it is not blank, it specifies the location at which the machine begins to assign the symbolic names that were not assigned by the program. (This machine assigning always starts with the address in the location counter after the END card has been processed.) On BRLESC, when this location field is not blank, all of the machine assigning will be done from that address.

Some examples of END cards are:

```
                    END GOTO(START)%
C3 + 1              END GOTO(1.1)%
```

M. DATE

This pseudo order type allows the current date to be punched in columns 11 - 20. This date information will be printed out in front of the running output and also in front of the dictionary when it is listed. On BRLESC, the date is obtained from the internal clock and DATE cards are ignored.

Example: DATE AUG 5,65

N. COMM

This pseudo order type allows columns 11 - 76 to be used for comments. The entire card is essentially ignored. (The location field is processed on BRLESC.)

O. MODE

This pseudo order type may be used to specify the type of arithmetic (fl. pt. or fixed pt. fraction) that is most used in the program that follows. It controls the type of arithmetic and number conversion in arithmetic formulas and DEC numbers that do not explicitly specify a different type of arithmetic.

If no MODE card is used, the mode is automatically set to floating point and thus a MODE card is not usually needed. To set the mode to fixed point fractional arithmetic, columns 11 - 76 of a MODE card should contain FIX(or just X). To set the mode back to floating point arithmetic, FLT(or just F) should be used in columns 11 - 76. Each MODE card is effective until the next one, however it generally isn't necessary or desirable to use many MODE cards. Note that the mode cannot be set for integer arithmetic.

The mode is not used on ORDVAC assembly orders, thus an ORDVAC assembly order is not floating point unless it begins with an F. (See page 108). The mode is used on BRLESC arithmetic assembly orders except the shift order. If a BRLESC A,S,M,D,C,SQRT, or PMA order does not explicitly have an F or X parameter specified, then the mode is used to set the order to the type of arithmetic that was specified on the last MODE card. A MODE designation of FU may be used on BRLESC to indicate "unnormalized" significant digit arithmetic. (An FU mode on ORDVAC is the same as FLT.)

The mode setting affects only the conversion of numbers on DEC (or DEC =) cards and the translation of the arithmetic operations in formulas. It does not change the type of subroutines used. Since practically all of the standard subroutines included in FORAST do normalized floating point arithmetic, new subroutines with new names must be added before subroutines can be used in fixed point or unnormalized floating point modes of arithmetic. It must also be remembered that exponentiation of the form A**B uses the floating point POWER subroutine whenever B is symbolic or is a non-integer number whose fractional part is not .5 or is an integer larger than 15.

The location field should not be used.

Examples:
```
MODE FLT
MODE FIX
```

P. STOR

This pseudo order type may be used to specify a location at which the following code should be stored. It uses the location field in the normal way or if it is blank, it uses columns 11 - 76 as a location field. Thus a STOR card may be used when one wishes to specify a location that requires more than six columns. A STOR card does not cause space to be left, putting a location on a STOR card is the same as putting it on the following card.

Examples:
```
      STOR  0800
      STOR  AA.5 + 17
B1    STOR
```

Q. NOS.

This card may be used to move the memory space that the translator uses for a "constant pool". Constants that are written in formulas (and other places where a store address is not provided) are stored in this "constant pool". This normally starts at 040 on ORDVAC and 0S0 on BRLESC. A different starting machine address may be specified in columns 11 - 76 of a NOS. card. This specification must be made before any constants are stored and should normally be done before the first order of the program. The "constant pool" cannot be moved very far. Since it must stay in the memory during translation, it cannot be put any place that would destroy a part of the translator. On BRLESC, it must stay between 0S0 and 020L or 01200 to 01800 and on ORDVAC it must stay between 040 and 029L. On ORDVAC, the maximum size of dictionary

allowed is reduced when the constant pool is moved down (larger address) in the memory. If the constant pool goes beyond OLL (or 01040 on BRLESC), then the location counter must be set so that it begins at an address that is larger than the usual 0100 (or 01040 for BRLESC) starting address.

Examples: NOS. 080
NOS. OLO

R. FTTS

This order type may be used to change the temporary storage that translated formulas will use in the running code. The ORDVAC normally uses 020-03L for this purpose and BRLESC uses 090-0KL. Columns 11 - 76 of a FTTS card may contain a machine address that will be used as the initial address of 32 words that may be used for temporary storage. (While 32 words are reserved for this purpose, most formulas do not require more than four or five words for temporary storage.) Each FTTS card is effective until the next one.

The location field should not be used.

FTTS cards are not needed in most programs. They are only needed when formulas are used to code a single valued function subroutine that will be used in other arithmetic formulas. In such a case, two FTTS cards are needed; one at the beginning of the subroutine to move the temporary storage away from its normal place and one at the end of the subroutine to set it back to normal.

Note that the same FTTS cards will not usually work for both ORDVAC and BRLESC.

Examples: FTTS 010
FTTS 020

S. ASGN

This pseudo order type is the same as SYN (See page 44) except there is no check for reassignment. It should be used only when it is necessary to change the assignment of a previously assigned symbolic address. This is the only way that an assignment can be changed.

Example: ASGN (A = 0800)(N = 3) %

T. SUBR

This pseudo order type is the same as ASGN except that the names used get marked as subroutines (except for single names). However it is recommended that SUBR be used only to define the names of single valued function subroutines that are coded as part of the program and are used in formulas in the main part of the program. Such names should be listed in columns 11 - 76 with each name enclosed in parentheses. A % causes the rest of the card to be ignored. These subroutine names must be at least three characters long and will not be marked as subroutines in the dictionary listing. The SUBR card must appear in the program before the names are used in formulas.

Example: SUBR (FIX)(F2X)GTW3 %

U. ALFN (BRLESC only, ignored by ORDVAC)

This pseudo order type allows alphanumeric constants to be stored. Columns 11-20 will be stored as ten six-bit characters in one BRLESC word or two ORDVAC words just as it is punched on the card. Each succeeding ten column field is stored in consecutive words if it is not all blank. If any ten column field is all blank, (beginning at column 21) it is not stored and the rest of the card is ignored. The last ten columns (71-80) are never used on ALFN cards. CONT cannot be used to continue a string

of alphabetic constants, each card must have an ALFN order type. Note that at least one (two on ORDVAC) alphanumeric word is always stored from each ALFN card, even if it is all blanks. Also note that some multiple of two words is always stored on ORDVAC.

Example: ALFN ERROR PRINT X = 0

V. O.T. (BRLESC only)

This allows arbitrary symbolic order types and their sexadecimal equivalents to be defined. Columns 11 - 76 may be used to define any number of order types. Each definition should be of the form: (symbolic order type = sexadecimal equivalent) and each definition should be enclosed in parentheses. After a %, the rest of the card is ignored. The symbolic order type must not contain any of the special characters and the sexadecimal equivalent may be written either with or without a leading zero. The primary purpose of having this pseudo order type is to allow the arbitrary definition of the interpretive orders on BRLESC, however it may be used to define any new order types or to define new names for old order types. (See page 136 for a list of BRLESC order types.) Note that an O.T. definition defines all eight bits of the order type. A maximum of twenty new order types may be defined in any one program.

Examples: O.T. (TI=LK)(AD=L2)%
O.T. (MM=L4)ADD=20)%

X. List of Standard FORAST Subroutines.

The FORAST compilers include about fifty standard subroutines that are available for use in any FORAST program. If the standard name of any of these subroutines is used anywhere in a program, the compiler automatically provides for storing that subroutine as a part of the running program. These subroutines are stored immediately before the decimal input-output routine and use as much space as

is required to store them all consecutively. (The "% SUBS" name at the end of the dictionary lists the address at which the subroutines begin.) A name of a subroutine should not be used as the name of a variable although this would work if the subroutine is not used in the program. The name of a subroutine actually represents the entry word which is usually the first word, however some subroutines have more than one entry and hence may have more than one name. Some subroutines also use other subroutines and the compiler always stores all the required subroutines but never stores the same one twice. The dictionary only shows the subroutine names that were explicitly used in a program.

All of the standard subroutines that do arithmetic do floating point arithmetic and hence must only be used when arguments are floating point numbers. New subroutines with new names would have to be added to do any other type of arithmetic.

The following floating subroutines have one argument and one result and may be used either in arithmetic expressions or in an ENTER statement:

SQRT	EXP	TAN	SINH
SIN	ARCTAN	COT	COSH
COS	ARCCOT	SEC	TANH
LOG	ARCSIN	CSC	WHOLE
LOG10	ARCCOS	SIGN	FRACT

ABS may also be used for absolute value but cannot be used in an ENTER statement. It may be used with any type of number but remember that BRLESC index registers will always be considered to be positive by an ABS command.

The angle arguments and results for the above subroutines are in radians. The result of ARCTAN and ARCSIN will be in quadrants I or IV and the result of ARCCOT or ARCCOS will be in quadrants I or II. Note that LOG is used for the natural logarithm and EXP is the e^x function. The SIGN function produces a result of -1(fl.pt.) when the argument is negative, a zero when the argument is zero, and +1(fl.pt.) when the argument is positive. SINH, COSH and TANH are hyperbolic functions. The WHOLE and FRACT subroutines produce the whole part or the fractional part of a floating point number as another floating point number without any rounding of the argument. The argument and result addresses are indexable on all of the above subroutines and the argument may be an arithmetic expression only when the subroutine is used in an arithmetic expression.

The rest of the subroutines listed here are normally entered with an ENTER statement. All of the addresses are indexable except where noted otherwise. A small letter means that the subroutine uses the effective address itself (not its contents) as an integer number. Underlining indicates those argument addresses that are optional and may be omitted. The addresses used here only illustrate the number and type of addresses required for each subroutine. In a program, they may be replaced with any other names or addresses.

SINCOS)X)SINX)COSX)p%	Assumes X in radians when p = 0 or is omitted. Assumes X in degrees when p = 1. Assumes X in mils when p = 2. Assumes X in circles when p = 3. (1 circle = 360 degrees.)
POWER)A)X)APX)%	Raises A to the X power where X may be either fl. pt. or integer number and A > 0 unless X is a whole number.
ARCSC)SINX)COSX)X)%	Computes X in radians from a known sin and cos. ($-\pi < X \leq \pi$)
ARTAN)Y)X)ANGLE %	Computes arctan (Y/X) in radians. ($-\pi \leq$ ANGLE $< \pi$)

WH.FRA)X)WHX)FRAX)%	Separates a number X into its whole part and its fractional part.
CVFTOX)FN)XN)%	Converts a fl. pt. number to fixed point fraction.
CVXTOF)VN)IN)%	Converts a fixed pt. fraction to a fl. pt. number.
CVFTOI)FN)IN)%	Converts a fl. pt. number to an integer after rounding by 10^{-4}.
CVITOF)IN)FN)%	Converts an integer to a fl. pt. number.
INTEGE)FN)IN)%	Same as CVFTOI
FLOAT)IN)FN)%	Same as CVITOF
READ BL %	Read a "blank" card.
PRINT B or PUNCH B%	Print a blank card.
SEXA PR)A)B)%	Sexadecimal print from A to B. ORDVAC: A and B are not indexable. A card with A and B on it is printed and then the contents from A to B are printed with eight words per card. Every word from A to B (inclusive) is printed. BRLESC: Prints the address of the first word on each card at the beginning of the card. Prints four words per card but does not print cards that would have all four words of zeros.
BCMPO)A)B)%	Binary card memory print out. Prints the contents of the memory from A to B on the binary cards with key words for re-reading by the standard program input routines.

ORDVAC: A and B are not indexable. Words of all 0 or all 1 bits are skipped except for the first word in a string of such words.

BRLESC: Words of all 0 bits are skipped except when there is only one zero word between non-zero words.

A.READ)A)n)%
A.PRINT or
A. PUNCH)A)n)%

Alphabetic card read and print routines. A is initial store address and n is no. of cards. Each card uses eight memory cells on BRLESC and sixteen cells on ORDVAC. BRLESC allows a third optional address "c" that is the number of columns per line for variable length line tape input or output. Each line starts at a new word and requires $\left\lceil \frac{c}{10} \right\rceil$ words.

The following subroutines may be used (in an ENTER statement) to change options in the decimal input-output routine.

ZEROCC	Sets the "card counter" to zero.
SETDPI	Set for double punched input signs.
SETDPO	Set for double punched output signs.
SETSPI	Set for single punched input signs.
SETSPO	Set for single punched output signs.
SETMSI	Set minus sign for input.
SETMSO	Set minus sign for output.
SETPSI	Set plus sign for input.
SETPSO	Set plus sign for output.

All are followed by one non-indexable addresses that is 0,1,2 or 3.
0 means blank
1 means y(12) punch
2 means x(11) punch
3 means x or y punch

NOTE: SETPSI doesn't do anything; all input numbers that are not negative are considered to be positive.

MAX.)A)B).....)Q)R%	MAX. finds the largest number of (A,B,....,Q) and stores it in R where A, B,...,Q are floating point numbers. Note that the number of addresses is variable (there must be three or more) and R is used here to denote the last address.
MIN.)A)B).....)Q)R%	MIN. finds the smallest number of (A,B,....,Q) and stores it in R where A,B,....,Q are floating point numbers.
MAX.I)I)J).....)K)R%	MAX.I finds the largest integer number of (I,J,.....,K) and stores it in R.
MIN.I)I)J).....)K)R%	MIN.I finds the smallest integer number of (I,J,.....,K) and stores it in R.
MOD.)A)B)C%	MOD. computes C = A(mod B) where A,B and C are floating point numbers. It is the same as the FORAST formula C = A - WHOLE(A/B)*B.

The following matrix manipulation routines are available:

S.N.E MAT.INV.	)A1,1)n)Co)DET%	A1,1: B1,1: C1,1 are addresses of the first elements of matrices.
	To omit the use of Co and use DET it is necessary to write A1,1)n))DET)	n is the number of unknowns (rows). Co is the address of the first element of the solution. DET is the address of the determinant.
SY.SNE SY.INV	)A1,1)n)Co)DET%	C1 is the address of the first coefficient of the given equation.
F.N.E.	)A1,1)n)C1)W%	i is the number of rows in A(or A^T).
F.O.MAT	)A1,1)n)%	j is the number of cols. in A(or A^T) and is equal to the no. of rows in B(or B^T).
MAT.M	)A1,1)B1,1)C1,1)i)j)k)% (No indexing on ORDVAC)	k is the number of columns in B(or B^T).

MAT.MP)A1,1)B1,1)C1,1)i)
j)k)z)ra)ca)
rb)cb)rc)cc%

NOTE: The augmented column is not counted.

z is 3 sexadecimal characters (preceded by a zero) that defines the options to be used in the matrix multiply (MAT.MP) subroutine. The sexadecimals correspond to the three matrices A,B,C respectively: 1 is transpose, 2 is augment, 4 is accumulate (Use 4 for C only; options may be combined; augment options ignored if r and c are specified.)

ra is the spacing between first elements of successive rows of matrix A.

ca is the spacing between the first elements of successive columns of matrix A.

rb,cb, and rc,cc have the same meaning as ra and ca except they apply to matrices B and C respectively.

NOTE: These r and c optional addresses cannot be omitted except omission by pairs from the end of the ENTER statement.

NOTE: When the transpose of a matrix is used, the dimensions of the transpose must be specified.

Additional comments on the above matrix subroutines: The S.N.E. (Solve normal equations) assumes all elements of a matrix having n rows and n + 1 columns are stored in the memory by rows. The SY.SNE (symmetric solve normal equations) assumes that only the upper triangle of an n x n + 1 matrix is stored and SY.INV (symmetric inversion) assumes that only the upper triangle of an n x n matrix is stored. S.N.E.; MAT.INV; SY.SNE; and SY.INV all replace the original matrix with its inverse. On ORDVAC only, the S.N.E. replaces the extra vector column with the solution vector besides storing it at Co if a Co is specified. The SY.SNE stores the solution vector only at Co. The F.N.E. (form normal equations) assumes that the upper triangular augmented matrix has been cleared by the program before it is entered with the first equation. The F.N.E. produces a matrix that can be solved with the SY.SNE. The F.O. MAT (fill out matrix) will take an augmented upper triangular matrix (as generated by F.N.E.) and replace it with an augmented square matrix (as needed by S.N.E.).

For BRLESC only, the S N.E. will attempt to rearrange rows of the matrix when it finds a zero diagonal element while it is computing the inverse. The row rearrangement does <u>not</u> affect the arrangement of the solution vector, however the inverse matrix will not be correct if any rows were actually rearranged. Rearrangement can be avoided by use of the "not" option as explained below.

Additional BRLESC S.N.E. options:

S.N.E.)A1,1)n)<u>Co</u>)<u>DET</u>)<u>drow</u>)<u>dcol</u>)<u>B1</u>)<u>db</u>)<u>dc</u>)<u>ZERO</u>)<u>not</u>)%

If "drow" is specified, it is the spacing between rows; i.e. the address A2,1 - address A1,1.

If "dcol" is specified, it is the spacing between columns (which is the same as spacing between elements within a row).

If B1 is specified, the n positions beginning at B1 are used as the column vector instead of the (n+1) column of the matrix.

If "db" is specified, it is the spacing between the elements of the column vector.

If "dc" is specified, it is the spacing between elements of the solution vector.

If ZERO is specified, it is the address of the number which will be used to check for zero diagonal elements. Those diagonal elements whose absolute value are less than ZERO will be considered as zero for the rearrangement test.

If "not" is any address different from zero, the S.N.E. will not rearrange any rows.

When any or all of these spacing options are omitted (or zero), the normal consecutive spacing of elements is assumed.

For MAT.INV on BRLESC, "drow", "dcol", ZERO, and "not" may be specified when needed and have the same meaning as for the S.N.E. except "not" has the opposite meaning. MAT.INV does <u>not</u> normally rearrange any rows and will do so only when "not" is specified as non-zero.

Note that when optional addresses are omitted any place except at the end of an ENTER statement, the right parenthesis must still be written for each omitted address. In particular, the above options for the MAT.INV subroutine must correspond to the same position on the list of addresses as used by the S.N.E. since they are just different entrance points to the same subroutine.

R.K.G.)Δt)n)Do)Yo)Ko)Qo)%
(Runga-Kutta-Gill solution of ordinary differential equations.)

R.K.GD is the return from the derivative sequence.
Use GOTO(R.K.GD)%

R.K.G1 is the re-entry for steps after initial entry.
Use GOTO(R.K.G1)%

Δt is the address of the step size.
n is the number of equations.
Do is the location of the first word of the derivative sequence. (must be a left order on ORDVAC)
Yo is the address of the first (independent) variable.
Ko is the address of the corresponding derivative.
Qo is the address of the corresponding error term.

RKGMA)Δt)n)Dō)Yō)Kō)KN %
(BRLESC only)

Use GOTO(RKGMAD)% at end of the derivative sequence.

Use GOTO(RKGMA1)% for re-entry at steps after the initial entry.

KN is a block of 3n memory cells needed for temporary storage within the subroutine. Integration by modified Adams method will automatically be started after four steps of Rung-Kutta-Gill integration. Equal step sizes are required. RKGMA must be re-entered for each change of step size.
If KN is omitted, entire integration will be by Runge-Kutta-Gill method.

S.INTE)Fo)T)FT)I)TL)TU)E %
(Simpson's Integration)

S.I.FF is the return from the function routine. Use GOTO(S.I.FF)%
(Note: The function routine must not be put immediately after the ENTER statement.)

Fo is the location of the first word of the function routine. (must be left order on ORDVAC).
T is the address of the independent variable.
FT is the address of the functional value at T.
I is the address of the integral.
TL is the address of the lower limit.
TU is the address of the upper limit.
E is the address of the relative error bound.

D.D.IN)X)FX)Xo)Fo)tpt)n)ix) if)%
(Divided Difference Interpolation)
Must use all three optional arguments or none. If omitted, (5)1)1) is used.

X is the address of the argument.
FX is the address of the result.
Xo is the initial address of the table of Xi's.
Fo is the initial address of the table of Fi's.
tpt is the number of entries in the table. (no. of Xi's)

D.D.SX)Fo)FX)%

Use this to interpolate more functions using the same value of X. (must use ENTER).

- n is the number of points to use in the interpolation.
- ix is the distance between entries in the X table.
- if is the distance between entries in the F table.

NRNOS1)Ao)n)Bo)%

(Normal Random no. generator)

NRNOS2)Ao)n)Bo)%

- Ao is initial address of n σ's (standard deviations).
- n is the number of random numbers desired.
- Bo is initial address of resulting random numbers.

(Use NRNOS1 for first entry in a program and NRNOS2 for all subsequent entries.)

NRNOS.)Ao)n)Bo %

(NRNOS. is faster than NRNOS1 on /ERLESC.)

ENTER(SETNRN)% may be used to reset NRNOS. to its initial sequence of random numbers.

G.L.SQ

or

P.L.SQ)X)ix)F)if)m)A1,1)n)C)R)ir)AT)iaf)ERMS)SIG)T)DET)w)iw)EQSEQ)TSEQ %

(General or polynomial least squares data fitting.)

- X For G.L.SQ, X is the location of the first term of the first equation. Terms must be stored consecutively. For P.L.SQ, X is the first independent variable.
- ix For G.L.SQ, ix is the distance from one equation to the next one. For P.L.SQ, ix is the distance from one independent variable X to the next one.
- F is the function value for the first equation or polynomial.
- if is the distance between function values.
- m is the actual total number of equations of "points" that are to be used in computing the fit. (It must not include those skipped by using EQSEQ.)

A1,1 is a block of storage that must be large enough for an augmented (n x n) symmetric matrix.

n For G.L.SQ, n is the actual number of terms to be used in each equation. (It must not include those skipped by using TSEQ.) For P.L SQ, n is one less than the number of terms and is the degree of the polynomial when all the terms are used.

C is the initial address for consecutively storing the n coefficients. (If $n \geq 38$, n + 1 spaces must be allowed at C.)

R is the initial address for storing the m residuals.

ir is the distance desired between residuals, i.e. the increment for the R address.

AF is the initial address for storing the m approximate function values.

iaf is the increment for the AF address.

ERMS is the store address for the root-mean-square error. Zero is stored when $m \leq n$ or when $\sum W_i \leq n$.

SIG is the initial address for consecutively storing the n "sigmas".

$$SIG_i = ERMS * SQRT(inv.el.A_{i,i})$$

(If the inverse element $A_{i,i}$ is negative, it is stored for SIG_i and $T_i = 0.$)

T is the initial address for consecutively storing the n "t's".

$$T_i = C_i/SIG_i$$

DET is the address to store the determinant.

W is the initial address of the weights to be used.

iw is the increment for the W address.

EQSEQ is the initial address of a consecutive sequence of numbers that have a one to one correspondence with each equation (or point) stored at X. A zero number indicates that the corresponding equation (or point) is to be used and a non-zero number indicates that it should not be used. Note that this sequence, if used, must contain m zero numbers.

TSEQ is the initial address of a consecutive sequence of numbers that have a one to one correspondence with the terms in each general equation or with the powers of X in a polynomial. A zero number indicates that the corresponding term or power of X should be used and a non-zero number indicates that it should not be used. Note that this sequence, if used, must contain n zero numbers.

For BRLESC only:

COWELL)Do)Δt)p)n)m)Yo)Ko)Qo)Vo)%

(Cowell's solution of 2nd order differential equations.)

where: Do; Δt; Yo; Ko; and Qo are the same as for R.K. G. except COWELL assumes the independent variable is in Yo and fi is in Yi and fi' is in Y_{i+n}.

p; n; and m are non-indexable.

p is the highest order difference to be used.

n is the number of functions to be differenced.

m indicates that 2^m steps of R.K.G. should be used for obtaining each of the p + 1 steps necessary for starting Cowell.

V_o is the initial address of a block of n(p+3) cells that the subroutine uses to store the difference tables.

The derivative sequence should end with a GOTO(COW.DX)% statement.

A "COWXTR" (Cowell extrapolation) subroutine is also available for use in satellite orbit calculations. See the separate description of this routine for details.

BESSEL)X)Jo % (BRLESC only) — where X is the argument and Jo is the initial address of six consecutive words for storing the three Bessel functions of the first kind of orders 0,1,and 2 followed by the three Bessel functions of the second kind of orders 0,1,and 2.

The following group of BRLESC subroutines simplify computations involving the real time clock.

R.CLK)CR %	Stores the current alphanumberic clock reading at CR.
CV.CLK)CR)FCR %	Converts the clock reading in CR to a floating point number and stores it in FCR. This number is the number of minutes since the previous midnight and is precise to hundredths of minutes. If the CR address is omitted (or zero), the current clock reading is used.
S.CLKS)CR1)CR2) DIF %	This subtracts two alphanumeric clock readings (CR2 - CR1) and stores the difference in minutes (modulo 24 hours) in DIF as a floating point number. If CR1 is omitted (or zero), the "start time" of the problem is used. If CR2 is omitted (or zero), the current time is used.
CK.CLK)MAXT)DONE%	If the total time in the problem has run up to this statement is greater or equal to MAXT (a floating point number) minutes, then the statement at DONE is done next. If DONE is omitted, N.PROB is used as that address. (Compile time is included in total time.)

The following group of subroutines allow easy use of magnetic tape input and output on BRLESC:

ENTER(SET.TI)u)E.T.)B)BMAX)v %

u	is tape unit number. ($1 \leq u \leq 5$ or $9 \leq u \leq 14$)
E.T.	is optional; if is zero (or blank) then the routine goes to N.PROB when the END TAPE sentinel is read. If specified (not zero), then the routine jumps to that address when the END TAPE sentinel is read.
B	is the initial address of a block of storage that is large enough to hold the largest block on the tape being read.
BMAX	is the last address in the storage block for this tape.

v — if v = 0, (or omitted) then computer does automatic selection between 80 character lines and variable length lines.
if v = 1, then tape is read only as 80 character lines.
if v = 2, then tape is read only as variable length lines.

SET.TI allows a program to read data on magnetic tape. It sets the computer so that subsequent READ statements (or A.READ or READBL subroutines) will cause data to be read from the tape unit specified. The tape may be one that was made off-line from cards or it may be a previous output tape. As many as six input tapes may be used in one program by entering this subroutine at different times with different tape unit numbers. If the unit number has been used previously in the program, the data will continue with the "line" that follows the last "line" that was read from that tape. Each unit should have its own storage block if the program ever re-uses that unit because a part of a block may need to stay there while another unit is being used. When entering with a unit that was previously used, it is not necessary to specify the storage block addresses; if specified, they will be ignored. It is not possible to change the storage block once it has been already assigned. The storage block may be longer than any block on the tape but must not be shorter than the longest block that is read from the tape. (Each storage word can hold ten characters.) The tape block length can be variable and if any block is longer than the storage allocated, the rest of the block will be ignored. All tape reading is parity checked and re-read five times before causing the erroneous "card" to be punched and a RUN ERROR card saying "PAR.ERRORu".

The "E.T." (end tape) address should be zero unless it is actually needed. If it is zero, this tape unit is rewound, the computer is set to read cards and control goes to N.PROB. If an address is specified, then these things should be done by the program before going to N. PROB.

It is desirable that a standard end of tape sentinel be used by everyone. It is also best to have a standard end of reel sentinel. This routine uses "ENDbTAPEbb" (b is blank) as the end of tape sentinel when it appears as the first ten characters at the beginning of a block and the next ten characters do not say "ENDbREELbb". When the next ten characters do say "ENDbREELbb", then it assumes that there is another reel to be read on this same unit, so it rewinds the tape and halts at 081 so that the operator can mount the new reel. (The unit no. is in the B address of the halt order.) Standard BRLESC tape 8 output will have the END TAPE sentinel if the "rewind tape 8" switch was properly used. When making tapes off-line, an extra block of one card with this sentinel should be added at the end of all the data.

When reading tape, the three "header cards" that were produced in front of previous FORAST or FORTRAN output are automatically skipped. (It checks for "bbBRLESCbb" in characters 11-20.) A dictionary or any other compiler output will also be automatically skipped.

When variable line tape is read, the vertical control character and any ignore characters are ignored. The "Δ" end-of-line character must appear to mark the end of the line but it is never stored as part of the data. Any line may be read as 160 characters long with blanks being used to expand the actual length to 160.

A word within this routine is named SKP.TL and it may be used to "skip tape lines". If it is set to an integer (not fl.pt.), then the next tape read will skip that many "lines". (If the skip includes the "header cards", then they must be included in the integer that is put into SKP.TL).

It is permissible to set for tape unit u when the same unit u is being used at the time SET.TI is entered.

ENTER(SET.CI)%

The input data may alternate between tape and cards at will. The use of SET.CI will set the computer for reading cards. (SET.CI is a small subroutine within the SET.TI subroutine. If tape No. 6 is being

used instead of card input, then it sets for tape 6 input. If the compiler is set for "card" input at the time SET.CI is entered, it does nothing.)

ENTER(SNB.TI)%

This subroutine will cause SET.TI to start a new tape block when it next uses the unit that is currently set for tape input. (SNB.TI does nothing if computer is set for card or tape 6 input.)

ENTER(SET.TO)u)B)BMAX)line %

This "set tape output" subroutine causes the subsequent PRINT or PUNCH statements (or A.PRINT or A.PUNCH or PRINT B subroutines) to write their output on magnetic tape. It allows switching between several output tape units, switching between tape and cards (which may be tape 8 at the operator's descretion), and also allows the line length to vary between one and 160. All tape output will have the necessary control characters for variable length line printing.

The parameters are:

u	Tape unit number. Must use 0 to 13 but not 6 or 7. Tape 8 (or 0) should be used if there is no card output intermixed with the tape output and if it is not necessary to have the output on a reel by itself. (If u = 0, tape 8 is used. Tape 8 is the standard unit used when getting "card" output on tape.)
B	Initial address of a "buffer" block for unit u.
BMAX	Final address of the "buffer" block for unit u. BMAX must be at least B + 7 (8 wds). The length of the block used is made mod 8 and not more than 200 words are actually used because of the 8K memory limitation of the 1401.

line — Maximum line length for unit u. ($1 \leq$ line $\leq$ 160 or 132 if ever want to print on 1401.) For initial entrance for each unit, zero is used as 80; afterwards, zero means "no change".

When switching is done between one or more tape units, each one must have its own buffer block and line length. Once a buffer block is specified for a unit, that same block is always used and it cannot be changed and it need not even be specified after the initial entrance. The line length is remembered for each unit and is reset when switching back to that unit. However it may be changed at anytime. (When changing the line length, the buffer addresses or at least the parentheses, must be written in the ENTER statement.)

The concept of one line completely replaces the concept of one 80 column card when the line length is changed. This means that the end of the format means start a new line, a format type 14 means start a new line, the end of a subgroup means start a new line, and it means that the card counter, when used, will print at the end of the line whenever the printing stops before the card counter field is reached. These concepts, except for card counter printing, are also true when reading variable length lines.

It is permissible to switch to the same unit that is currently being used. A maximum of six tape units (besides tape 8 "card" output) is permitted.

When using unit 8, it is not necessary to specify a buffer block. If none is specified, the standard buffer (at ONF60-ONL27) is used. (This is also used for "switch 35" output.)

All tape writing is parity checked and concurrency is gained by "double-buffering". ONL30-ONLL7 is used as the extra 200 word buffer. Tape trunk B is used to do all of the writing.

The end of a reel is checked for and when reached, the "END REEL" sentinel is written, the tape rewound and the computer halts at 080 with the unit no. in the B address of the halt order. Mounting a new tape (or changing the proper tape switch) and initiating is all that is required of the operator to continue writing on a new reel.

ENTER(SET.CO)%

This SET.CO subroutine (actually part of SET.TO) allows switching from tape output to card output (or tape 8 if switch 35 is up). It resets the line length to 80. It does nothing if the computer is already set for "card" output.

ENTER(EBR.TO)u)r %

This subroutine (actually part of SET.TO) will empty the buffer block used for unit u by writing a partial block on the tape. If r is omitted or zero and u is not 8, it will also write a file mark, write an 8 word END TAPE block, another file mark and rewind the tape. If r is not zero or if u is 8 and tape 8 is being used for "card" output it will only write the partial block.

EBR.TO may be used with the current unit, any previously used unit or if the unit has never been used, it does nothing. It also does nothing if the tape has just been rewound by this subroutine. It may be used with u = 8(or 0) to empty the "switch 35" tape buffer and it will do so only when switch 35 is up. It will not rewind tape 8 when switch 35 is up regardless of the r used.

Normally, the programmer does not need to use EBR.TO as the N.PROB routine will empty the buffers and rewind all of the units that have been used by SET.TO. (It does this by actually using the EBR.TO subroutine.) If C.PROB is actually used instead of N.PROB, the buffers are emptied but the tapes are not rewound. Any N.PROB (or C.PROB) error prints will appear on the tape (or "cards") that was being used at the time N.PROB (or C.PROB) was entered. If the card deck version of N.PROB is used by the BRLESC operator, the error prints will be on cards and the tape buffers will not be emptied and the tapes will not be rewound.

Once EBR.TO has been used, the buffer block may be used for other storage. However, if more output on that unit is desired, the first word must be set to contain the sexadecimal constant (ZLN1). (This is an ignore character and a 1 character.)

Variable Length Line Control Characters:

The SET.TO subroutine and tape 8 "card" output both write variable line control characters as described for the off-line printer system. Thus each line has a vertical control character at its beginning that controls the amount the paper is shifted and a "Δ" end-of-line character at its end.

The vertical control character is not printed. Its rightmost four bits selects a paper tape channel and the printer carriage moves the paper until it finds a hole in the paper tape in the selected channel. It is extremely desirable that everyone uses the same standard paper tape which defines the control characters in the following manner:

CONTROL CHARACTER	DESCRIPTION
1	Single space and start a new page after 60 lines.
2	Skip to next even numbered paper line and start a new page after line 62 has been printed. (Is double spacing if used all the time.)
3	Single space continuous.
4	Skip to next 1/5 page. (12 lines per group.)
5	Skip to next 1/4 page. (15 lines per group.)
6	Skip to next 1/3 page. (20 lines per group.)
7	Skip to next 1/2 page. (30 lines per group.)
8	Start new page at line 3.

ENTER(SET.VC)n)s %

SET.VC is a subroutine that allows the programmer to specify the vertical control character for the next line and also the succeeding lines. The n address itself is the control character for the next line

and the s address is the control character for all succeeding lines, (until SET.VC is used again.) If either (or both) address in blank (or zero), it is ignored and the previous character remains in effect. The n character is used for the next tape output line regardless of whether it is tape 8 "card" output or SET.TO tape output.

Both n and s are initially set to 1 (single space).

To use the selective printing option available on the Analex system, the decimal or sexadecimal equivalent of the desired six bit code may be written for both n and s.

Example: ENTER(SET.VC)8)2% means start a new page with the next line and double space the succeeding lines.

BRLESC Compiler Tape Output:

All compiler output (error prints, header cards, dictionary and sexadecimal code) may be put on tape 8 either by the computer operator putting switch 35 up or by a SET.TO % statement appearing in the program being compiled. (Note the absence of ENTER in front of SET.TO!) All compiler output may be put on another tape unit u by writing the statement SET.TO(u)% in the program being compiled. Note that SET.TO% and SET.TO(u)% statement (without ENTER) set the computer for tape output at the time they are being recognized by the compiler and not when encountered in the running program. They both also cause the running program "card" output to be put on the same tape unit. However SET.TO(u)% may actually be changed to card output by using ENTER(SET.CO)% in the program if $u \neq 8$.

All compiler tape output is written as variable length lines with a new page begun at the first line of output and at the header cards in front of the problem output.

BRLESC Tape Plotting Subroutine:

BRLESC has a subroutine, with eight entrance names, that can be used to produce magnetic tape for off-line plotting of data. The names of the subroutine entrances and the lists of addresses used with them are listed below without explanation. See BRL Technical Note No. 1551 for a detailed description of this subroutine.

PLOT.S)h)$\bar{x}$)$\bar{y}$)X3)Y3)XS)YS)Ah)u %

PLOT.D)h)f)a)an)X)Y)n)ix)iy)c)XMIN)XMAX)YMIN)YMAX %

PLOT.A)h)ΔX)ΔY)XMIN)XMAX)YMIN)YMAX)an %

PLOT.T)h)HT)SY)p)an)X)Y %

PLOT.F)h %

ENDPLO)b)h %

FIX.SC)D1)n)i_D).5IN)DS)DMIN)DMAX)DELD %

CON.SC)D1)n)i_D).5IN)DS)DMIN)DMAX)DELD %

Additional information on most of the above subroutines may be obtained from their individual descriptions that are available.

XI. Error Prints

Both FORAST translators do a limited amount of checking for programming or key punching errors. They check for only very simple types of errors and this is only a very small percentage of all possible errors. The programmer must not assume that his program is correct just because it does not cause an error print. If the translator does find an error, it prints out some information about the type and location of the error. The ORDVAC prints two words on the teletypewriter and BRLESC prints two alphanumeric cards. When a program contains an error that causes an error print, the translators will always print the dictionary and will not allow the program to be run.

When the translators find an error, the rest of the card that contains the error is not translated. If the error is found at the end of a card; the translators will usually skip the entire next card. Therefore it is possible that the translators will not find all of the errors in just one run.

The dictionary should be used to help locate errors. It should be checked for strange names that will usually be marked with a U and for the desired storage assignment. The dictionary does not need to be checked or printed each time a problem is translated, but it should be checked at least once for a given problem.

A. ORDVAC Error Prints:

The ORDVAC prints two words on the teletypewriter that have the following form:

Type 0 0 m 0 Ident. Card No.

where Type is the two digit number listed below; m is the sexadecimal digit (0 to L) that indicates the five digit field on the card in which the error was found; and Ident. is the decimal part of columns 77-80 that were on the card that contains the error. (The error might be at the end of the previous card if m is zero.) The second word printed is the actual sexadecimal count of the number of cards that have been processed. It is independent of the identification in columns 77-80 and is strictly a count of the number of cards that have been "read" at the time of the error print. It starts at one and if a five is printed and $m \neq 0$, the error is probably on the fifth card in the program.

Below is a list of the ORDVAC type numbers with a brief description of the probable cause for the error print.

TYPE	PROBABLE CAUSE
01	Illegal order type in assembly language.
02	Illegal character.
03	More than one comma in a non-array address.
04	Illegal character in a decimal number.
06	Illegal character in a sexadecimal word or address.
07	Illegal exponentiation.
08	IF statement has ABS in front of an equality conditional expression.
10	Symbol after ")IS" is neither + nor - in an IF statement.
11	Illegal name preceding a conditional expression in an IF statement.
12	Improper name at the beginning of a formula or statement.
13	IN does not appear in a COUNT statement.
14	No % at end of COUNT, MOVE and CLEAR.
15	No % at end of a MACH assembly order.

TYPE	PROBABLE CAUSE
16	Attempted to reassign an address.
17	Referenced a block before it was defined.
19	Format address is indexed in a READ, PRINT or PUNCH statement.
20	Increment for address advance or count in a COUNT, CLEAR, MOVE, READ or PRINT statement has an indexed symbolic primary address.
21	Fixed fractional number exceeds one.
22	The address on FTTS or NOS. card is not a machine address.
23	The GOTO address on the END card is not assigned.
24	An address has a symbolic increment.
25	Name of the subroutine is indexed in an ENTER statement.
26	Address on LAST card becomes assigned before the END card.
30	Illegal arithmetic on left of = character in arithmetic formula.
60	Symbolic name used for no. of blanks before > in a PRINT statement.
61	No > on the same card after a < in a PRINT statement.
F1	Dictionary is full. (About 380 names.)
F2	The SYN backlog table is full. (28 names.)
F3	The drum is full. (Program is too long. About 3000 words.)
F4	The SYN table is full of unassigned addresses. (64 names.)
J4 040	Exceeds drum capacity while reading cards.
NO FNJ	No END card.

B. BRLESC Error Prints.

BRLESC prints two alphanumeric cards. The first one is of the following form:

ERROR Type Error word m Ident. Rest of Pm cols. 11-20 Prob. card

where Type is the two character number listed below; Error word is a ten character word that attempts to describe the type of error; m is a digit (0 - 7) that indicates the ten character field in which the error was found; Ident. is columns 77 - 80 of the card that contains the error (or

of the next card); "Rest of Pm" is the characters that have not been used from the mth field on the card (the error was found at the leading character printed here); Columns 11-20 from the card are printed; and "Prob. card" is the first 30 columns from the PROB card that was at the beginning of the program.

The second card printed is the acutal card that contains the error or when m = 0, it is probably the card following the one that had the error.

Below is a list of the BRLESC type numbers and error words with a brief description of the probable cause for the error print.

TYPE	ERROR WORD	PROBABLE CAUSE
01	ILL. O.T.	Illegal order type or statement.
02	ILL. CHAR	Illegal character.
03	2 COMMAS	Two commas in a non-array address.
04	DEC NO NO.	A decimal number has illegal character.
06	NOT SEXA.	A non-sexadecimal character in a sexadecimal word or address.
07	ILL. **	Illegal exponentiation.
08	ABS IN IF =	IF statement has ABS in front of an equality conditional expression.
10	NOT IS+OR-	Symbol after ")IS" in IF statement is neither + or -.
11	ILL PAR IF	Illegal "parameter" in front of conditional expression in IF statement.
13	COUNT NOIN	A COUNT Statement has a variable increment without using IN.
14	COUNT NO %	No % at end of COUNT statement.
14	CL MV NO %	No % at end of CLEAR or MOVE statement.
15	NO % AS.	No % at end of an Assembly order.
16	SYMB. RESN	Attempted to reassign a symbolic address.

TYPE	ERROR WORD	PROBABLE CAUSE
17	BLOC REF.	Referenced a block before it was defined.
21	X NO. OVER	A fixed pt. fraction exceeds sixteen.
22	NO MAC ADD	No machine address on FITS or NOS. card.
23	END SYMB.	The GOTO address on the END card is not assigned.
24	SYM. INC	An address has a symbolic increment.
26	LAST SYMB.	An address on a LAST card becomes assigned before the END card.
30	ILL. LOC.	Improper location field.
31	FORM. BIG	Formula is too big; has more than 31 operations grouped from the right.
32	ILL. OP. c	Illegal operation where c is the illegal symbol.
33	INT. SQRT	Integer square root is not allowed.
34	NO GOTO IF	An IF statement without a GOTO address.
35	ILL. < OR >	Illegal inequality symbol. (not in an IF statement.)
36	WITHIN NO=	WITHIN used following an inequality conditional expression.
37	NO % SI GO	A GOTO in a SET or INC statement is not at the end of the statement.
38	NO % SI II	A SI,II,SIJ, etc. assembly order has more than three addresses.
39	INDX SY IC	A symbolic increment for COUNT, CLEAR, etc. is also indexed.
40	HALT NO %	Have more than three addresses in a HALT order.
41	GOTO IF %	A GOTO statement is not followed by IF or %
42	ILL. O.T.	The order type name on an O.T. card is not symbolic.
43	ILL. C IO	Illegal C address on input-output order

TYPE	ERROR WORD	PROBABLE CAUSE
45	ILL.=	Illegal arithmetic on left of = character in arithmetic formula.
48	ILL.SHIFT	Illegal shift code character or an illegal fl. pt. shift.
49	2R. WITHIN	More than one relation preceded WITHIN in IF statement.
50	INDEX > 63	Used more than 54 index registers.
51	LOC = 000	Location counter is set to 000.
52	BLOC = 000	A block is assigned to 000.
55	NO PROB C.	No PROBlem card.
60	PRINT SYM >	Symbolic name for number of blanks before > in a PRINT statement.
61	PRINT NO >	No > on the same card after a < in a PRINT statement.
62	LOC. > _ Add._	Program is stored at too large an address. "Add." is largest allowable program address and changes with memory size. For 52K, it is 9LLL.
63	TAPE 7 NG	Repeated failures on temporary tape unit 7.
F1	DICT. FULL	The dictionary is full. (about 7000 names)
F2	BACKLOG	The SYN backlog table is full. (64 names)
F4	SYN. TABLE	The SYN table is full. (288 names)
F5	NOS. FULL	The constant pool is full. (352 constants)
	ERROR TAPE 7	Machine error on temporary tape. (Columns 69-80 has alphabetic error bits or a name that came from tape but is not in the dictionary.
ME	MACH. ERROR	Machine error.

XII. Run Error Prints:

Many of the FORAST subroutines include checks for error conditions at run time. If a subroutine finds an error condition, it causes an error print. After the print, the computer halts unless "ERROR" was used as a location in the symbolic program. If ERROR was used, then the program continues running at the ERROR location. (See section XIII;A).

A. ORDVAC:

The ORDVAC run error print routine prints two sexadecimal words on the teletypewriter. The first seven sexadecimal digits of the first word serve as a code number to identify the subroutine and the last three digits of the first word are the return address for the entry that caused the error condition. The second word is generally the argument or some other word that indicates the type of trouble.

ORDVAC Run Error List:

(X and Y represent arguments and RA is return address)

ERROR WORD	SUBROUTINE	REASON	SECOND WORD
K6012 NO _ RA_	SQRT	$X < 0$	X
06014 NO _ RA_	LOG or LOG10	$X \leq 0$	X
K6013 NO _ RA_	EXP	$X \geq 87.3365444$	X
26010 NO _ RA_	ARCSIN or ARCCOS or ARCSC	$\|X\| > 1 + 2^{-28}$	$\|X\|$
K6010 NO _ RA_	SIN	$\|X\|/2\pi > 2^{30} - 1$	$X/2\pi$
K6011 NO _ RA_	COS or SINCOS	$\|X\|/2\pi > 2^{30} - 1$	$X/2\pi$
12000 NO _ RA_	POWER	$X = 0$ and $y = 0$ or $X < 0$ and Y not an integer	-Y X
12000 KO _ RA_	ARTAN	$X = Y = 0$	Zero
7K011 KF _ RA_	TAN or COT or SEC or CSC	|Result| = ∞	X
08001 NO _ RA_	SINH or COSH	|Result| = ∞	Zero
10000 NO _ RA_	CVFTOI	$\|X\| > 2^{30} - 1$	X
10000 KO _ RA_	CVFTOX	Exp. of $X \geq 2$	Exp. - 2
74SN1 NO _ RA_	SY.SNE or SY.INV or S.N.E. or MAT.INV.	ith diagonal is 0	$(i-1)2^{-39}$
LN _ FA_ NO F6K	READ or PRINT or PUNCH	Unused format type. (FA is format address) or input no. is too big.	3N _ I _ K4 _ NA_ (NA,I is add. of next no.)

ERROR WORD	SUBROUTINE	REASON	SECOND WORD
88 _ _ _ NO _ RA_	D.D.IN	Either X is too far outside table or table is too small.	X
12 _SA _ SN _ _RA	S.INTE (SA is store add.)	Error limit is too small or range too large.	Rel. error at last step.

After the teletype error print, the ORDVAC also prints a card that says "SUB ERROR". Then if ERROR was not used, it halts at OLN7 and if ERROR was used, it continues to run the program by going to the location assigned to ERROR without stopping.

Since some subroutines use other subroutines, it is possible that the return address (RA) is a location in some subroutine rather than in the FORAST program.

B. BRLESC

The BRLESC run error print routine prints two cards, an alphabetic card and a binary card that contain the same information. The alphabetic card printed has the following form:

RUN ERROR Error word PROB.or Date Cols. 11-40 of PROB card LE(dec.) No.

where "Error word" is an alphabetic word that identifies the subroutine and tries to indicate the type of trouble (see list below); LE is the location of the entry to the subroutine (it is printed in decimal here, it can be read in sexadecimal on the binary card); and "No." is a decimal fl. pt. or integer number that is usually the argument that caused the error condition.

The binary card printed by BRLESC for a run error contains the same information as above but in a different sequence. The 12 row of the card (only columns 1 - 68 are used) contains LE, the 11 row contains "No." and the 0 to 5 rows contain the same alphabetic words that are printed in columns 1 - 60 of the alphabetic card described above. (The 1 row contains the "Error word".)

BRLESC Run Error List:

(X and Y represent arguments)

ERROR WORD	SUBROUTINE	REASON	NO.
LOG X NEG.	LOG or LOG10	$X \leq 0$	X
EXP BIG X	EXP	$X > 354.89$	$X/\text{Log}_e 2$
ARCSIN 1 +	ARCSIN or ARCCOS or ARCSC	$\|X\| > 1 + 2^{-49}$	$\|X\|$
SINCOS N S	SIN or COS or SINCOS	$\|X\|/2\pi \geq 16^{13}$	$X/2\pi$
POWER oTO-	POWER	$X = 0$ and $Y \leq 0$	Zero
TAN DIV. 0	TAN or COT or SEC or CSC	$\|\text{Result}\| = \infty$	X
SINH BIG	SINH or COSH	$\|\text{Result}\| = \infty$	Zero
CVFTOI BIG	CVFTOI	$\|X\| \geq 16^{14}$	X
CVFTOX BIG	CVFTOX	$\|X\| \geq 16$ or $X < -16$	X
S.INTEGRAT	S.INTE	Error limit too small or range too large.	Rel. error at last step.
SING.MAT	SY.SNE or SY.INV or S.N.E. or MAT.INV	ith diagonal is 0	$(i-1)2^{-60}$
D.D.IN	D.D.IN	Either X is too far outside table or table is too small.	X

ERROR WORD	SUBROUTINE	REASON	NO.
SET.TI > 6	SET.TI	Used more than 6 tape units.	New tape no.
SET.TI-BUL	SET.TI	Improper buffer specification.	New Tape no.
PAR.ERRORu	SET.TI	Parity error on unit u.	Not significant. 80 column error line is printed before error print.
T.O. u PE	SET.TO	Parity error on unit u.	Total no. of tape write errors.
SET.TO > 6	SET.TO	Used more than 6 tape units.	Zero.
TO.BAD BUF	SET.TO	Improper buffer specification.	Zero.
O.LINE > 160	SET.TO	Line length specified was > 160.	Zero.
E.P. NO HND	END PLOT	Improper tape unit no.	Tape unit no.
P.F. NO HND	PLOT.F	Improper tape unit no.	Tape unit no.
P.A. NO HND	PLOT.A	Improper tape unit no.	Tape unit no.
P.T. NO HND	PLOT.T	Improper tape unit no.	Tape unit no.
P.D. NO HND	PLOT.D	Improper tape unit no.	Tape unit no.
P.S. > 4 HND	PLOT.S	Used more than 4 tapes.	Tape unit no.
P.S. NO BUF	PLOT.S	No buffer specified	Tape unit no.
P.A. DELTA	PLOT.A	$\Delta X = 0$ or $\Delta Y = 0$	Tape unit no.

BRLESC halts at ON40 after an error print if ERROR was not used. If ERROR was used, it goes to that location without stopping and continues running the program.

The following BRLESC run error prints are obtained only when a problem goes to the N.PROB subroutine when it is finished. (A program that stops by reading more cards will go to N.PROB if it has a PROB card at the end of its data.) These error prints do not tell you where the error occurred, there is no LE or NO. printed. (See Section XIII;B)

ERROR WORD	SUBROUTINE	REASON
NEG. SQRT	N.PROB	Square root of negative number.
FLT DIV 0	N.PROB	Fl. pt. divide by zero.
FLT OV.FL.	N.PROB	Fl. pt. exponent overflow. ($X > 16^{128}$)
SHX OV.FL.	N.PROB	Fixed pt. shift overflow.
DX OV.FL.	N.PROB	Fixed pt. divide overflow.
MX OV.FL.	N.PROB	Fixed pt. mult. overflow.
ASX OV.FL.	N.PROB	Fixed pt. add or subtract overflow.
TAPE ERR.A	Tape trunk A parity error light is on.	
TAPE ERR.B	Tape trunk B parity error light is on.	

XIII. Use of Some Special Names.

A: ERROR

The name ERROR may be used by the programmer to specify a location to go to when a subroutine finds an error condition during run time. (See Section XII.) The computer halts after a run error print if ERROR has not been used (or if it has been used but not assigned an address

by the programmer). If ERROR is used (in a location field or otherwise assigned by the programmer), then the program continues running without stopping from that location after a run error print.

If all of the data for one "case" of a problem is read before it can encounter any error prints (other than READ error prints), then it is wise to use ERROR as the location for starting a new case. This allows a program to run or at least attempt to run all of the cases rather than stopping at the first case that won't run.

B. N.PROB

The name N.PROB represents "next problem" and a program should go there whenever it knows that it is finished running. A GOTO (N.PROB)$ statement should be used instead of a HALT statement in this situation.

On ORDVAC, N.PROB is assigned to OLN7 and is nothing more than a standard halt order. It has the advantage that the technician running the program knows that the problem ran to an expected halt. If a program halts elsewhere, he isn't sure whether it was expected or not.

On BRLESC, FORAST allows several programs to be stacked on tape and run consecutively and a GOTO (N.PROB) means exactly that. N.PROB is actually a subroutine on BRLESC that checks for overflow conditions (See page105 for overflow prints), empties tape output buffers, rewinds output tapes, and halts at N40. When programs are stacked on tape, they must be separated by file marks because the compiler always moves the input tape to a file mark before beginning compilation.

C: C.PROB

The name C.PROB means "compile next problem" and this allows a programmer to compile a problem, run it, compile another problem, run it, etc. It essentially allows a programmer to stack several problems together and run them as one problem. The deck will appear to be only one problem to the computer operator, the computer will not halt between each compilation and all of the time will be charged to one problem number.

Either ENTER or GOTO may be used to go to C.PROB which is really a subroutine that performs the duties of N.PROB except for rewinding the output tapes and halting.

When using C.PROB, each program must have a PROB card at the end of its data. C.PROB should never be used unless there actually is another program stacked behind the one in which it is used. The last problem in such a stack of programs should go to N.PROB.

If while reading data, the PROB card is read, transfer to C.PROB is done if C.PROB was mentioned anywhere in the program. However after a "RUN ERROR" print, control always goes to N.PROB and not C.PROB unless ERROR is used to specify where it should go.

D. M.DUMP

The name M.DUMP is designed for used as a code checking aid. It may be used as a location to go to after a program comes to an unexpected halt. If the person running a program manually transfers to 006 on ORDVAC or 058 on BRLESC when an unexpected halt occurs, (as he should), then the program will jump to and do the code at M.DUMP and this code can be used to print some information that might be useful in determining the reason for the unexpected halt. As the name implies a portion of the memory may be "dumped" to allow inspection by the programmer. Inasmuch as 006 (or 058) will execute a GOTO(N.PROB) order if no M.DUMP occures, the code at M.DUMP should always end with a GOTO (N.PROB)% statement.

XIV. Machine Assembly Language.

Each FORAST translator program allows its own machine assembly language to be used and assembly orders may be interspersed with formulas and statements. It is also possible to include both ORDVAC and BRLESC assembly orders in the same program in a manner that causes each machine to ignore the assembly orders that were intended for the other machine. (This is done by using MACH cards on ORDVAC, as explained below and by beginning each card of BRLESC assembly orders with a statement of "BRLESC %".)

Any order type or address in an assembly order may be either symbolic, sexadecimal or decimal. Actual numbers, rather than the address of a number, may be written in place of an address in assembly orders by preceding the number with the symbol*. Any type of number allowed in DEC (See page 56) may be written after the *. In the absence of F or X at the beginning of the number, the type of conversion is controlled by the type of order rather than the MODE. On ORDVAC, numbers in floating point orders are converted to floating point (unless X,I,B, or D is used) and numbers in fixed point orders are converted to fixed point fractions if they have a decimal point and to integers if they do not have a decimal point. On BRLESC arithmetic orders (except shift), the actual F or X parameter bit in the order determines the type of conversion. On non-arithmetic orders, the MODE determines the conversion. (Note that on BRLESC the MODE actually determines the F or X parameter bit on arithmetic orders unless F or X is written as part of the order type.)

A. ORDVAC Assembly Language.

ORDVAC assembly orders may be written one per card by writing the order type in columns 7-10 and the address in columns 11-76 or several may be written on the same card by using MACH in columns 7-10 and using columns 11-76 to write any number of assembly orders. BRLESC will ignore MACH cards but will attempt to translate the one order per card order improperly. Hence MACH cards should be used if the program is to be run on both machines.

On MACH cards, the address of each order is separated from the order type by a (and each order is separated from the next one by %. %% causes the rest of the card to be ignored. The enclosing) after each address is optional and the entire address may be omitted.

The address in an order may be indexed (by using a comma) and causes the translator to code an index order before coding the order that contains the indexed address. The index order may also be written separately.

Full word orders (drum and magnetic tape) must be written as two half words and the location field must be used to insure that both halves will be located in the same word. Note that the right address of full word orders cannot be indexed.

A blank (or omitted) address is the same as 000 except on the A series of orders. A blank address on any of the A orders is replaced by the address of the next word (SELF + 1).

U or GOTO and C or C+ transfer of control orders will be changed by the translator to transfer control to the right side if the primary address in the order is the location of a right order. Thus GOTO (3.1)% will transfer to 3.1 regardless of whether it is a left or right order. All absolute decimal, sexadecimal, or blank addresses are considered to be left locations for this purpose. SELF is also a left location.

The ORDVAC translator automatically leaves a blank half word when necessary so that DEC, SEXA, FORM, or DEC = constants will be stored in full words. When the translator needs to skip a half word before it codes the next order, it automatically inserts a Zx(SELF + 1) order.

Special processing is done on the address of the "tape select" (TS) order. The address may be written as R,W, MF, MB, REW, or UNW followed by a dash and an absolute tape unit number. The move orders (MF and MB) may contain a "second address" that it an absolute number of the number

of blocks the tape should move. For example, TS(MB-1)3% instructs tape unit 1 to move backward 3 blocks. Instead of one of these special symbolic names, the address may be an ordinary symbolic or absolute address.

The table of all symbolic order types for ORDVAC is listed in Appendix A. See [1] for a complete description of the ORDVAC and its instruction repertoire.

If an ORDVAC symbolic order type is written in columns 7-10, it must not contain any blank columns between any two characters of the order type.

A sexadecimal order type on ORDVAC sets all eight bits, including the spare bit. However a decimal order type sets only the leading six bits and does not affect the floating point bit or spare bit.

Examples of ORDVAC Assembly Orders:

```
        +       A
        (+)     X,I
        M       B,I %  Comment
        U       14.2
        MACH    +(A)%(+)(X,I)%M(B,I%%  COMMENT
        MACH    F+(T)%F/(V,J3)%A+%FM(R,J3)%
LEFT    MACH    DR(Ao)% 064(18.2)%
        MACH    RS( )%% MACH cards are ignored on
                BRLESC.
```

B. BRLESC Assembly Language.

BRLESC assembly orders may be interspersed at will with arithmetic formulas and other statements. If the program will not be run on ORDVAC, the order type of the first assembly order on

a card may be in columns 7 - 10. If the program will be run on ORDVAC, the statement "BRLESC %" should precede the first assembly order on each card.

Since BRLESC is a three address computer, the general form of assembly orders should be O.T.(A)(B)(C)% where (after a) is optional. It is permissible to write just O.T.% or O.T.(A)% or O.T.(A)(B)% where A and B are used to indicate the first and second addresses of a BRLESC order. However there are a few exceptions where it is not necessary to write all three addresses in order to have one used as a C address. SIJ or SETJ and IIJ or INCJ always use the last address as the C address. A HALT order (which is also a statement, (See page 36) uses the last address as a C address if there is more than one address written. If one address or no address is written in a HALT order, the C address is set to (SELF + 1) automatically. GOTO is permitted as an assembly order that has only one address.

The BRLESC symbolic order types are listed in Appendix B along with the symbolic parameters. See [2] for a description of the BRLESC instruction repertoire. The F or X (floating or fixed pt.) parameter may be written either before or after the order type while all other parameters must be written after the order type but may be written in any sequence. If F or X (or U or N or R) is not specified in a symbolic arithmetic order (except shift), the order will be set to do the MODE (See page69) arithmetic. Shift orders are always fixed point unless they are specified explicitly as floating point. The U or N parameter also sets the F parameter and R also sets the X parameter. If the MODE is FU (unnormalized fl. pt.), the U parameter is not set unless the order is actually a floating point order. Any symbolic order type begun in columns 7 - 10 must not continue into column 11. No symbolic order type may have more than six consecutive letters and/or decimal digits. (A minus sign may be inserted in front of the parameters.)

The entire order type (8 bits) may be specified in sexadecimal or decimal (no signs allowed). (MODE does not affect sexadecimal or decimal order types.) A symbolic order type followed by a decimal or sexadecimal (with or without a leading zero) parameter is allowed only on B,CB,CNB or CN, TP and IT orders. (Symbolic parameters are not allowed on these orders). For all other orders, the order type and parameter must be all symbolic or all decimal or sexadecimal.

Symbolic parameters must not be written without a symbolic order type. If a blank or zero order type is desired, a Z may be written for it and may be followed by any of the symbolic parameters.

Order types of CEQ and CNEQ may be used instead of CB6 and CNB6 respectively.

Key words for the standard program input routine may be written by writing an order type of KEYi where i is a decimal digit (0,1,2,3) that defines the type of key word. Key words are only required when a program is to be punched for use as a subroutine. Key words do not affect the storing or translation of a program and mean nothing unless a B.SUBR print is performed.

There are four BRLESC orders that have addresses that are not used as memory addresses. They are: (1) B address of SH, (2) B address of JA and JNA, (3) B address of JC and JNC, (4) C address of IO. FORAST allows these special "addresses" to be written in a special way.

The B address of a shift order is a shift code rather than an address. The type of shift may be specified symbolically by using the desired letters listed in Appendix B for the shift code. These letters may be followed by the magnitude of the shift in decimal

or sexadecimal and the address may be indexed in the normal way. The direction of shift should immediately precede the amount of shift. Note that R has a double meaning, it means round only if it is to the left of the direction (L or R or + or -) of shift, otherwise it indicates a right shift. The address assigned to a symbolic name may be used for the type and amount of shift by preceding it with "S/".

The B address of a JA or JNA order is used as a six bit character rather than as a memory address. Therefore FORAST allows a single character to be written and its six bit representation used as the B address of a JA or JNA order. If there is a (at the beginning of this address, it is ignored but only one (will be ignored. If the "address" is a special character or a single letter or digit, its six bit representation will be used. If the "address" has more than one character and is decimal or sexadecimal, its rightmost six bits will be used. If a symbolic name has more than one character or if it has an increment, the assigned address will be used except for the word BLANK which may be used to indicate a zero address. (The six bit representation of the blank character is six zero bits). Note that it is not possible to leave a blank address here as the) would be used as the character to be represented in the address. Note also that a single zero character will give a binary address of 11 0000 (the zero character) while 00 (two zeros) will give an all zero address and represents a "blank" character. It is permissible to use the symbol ' = ' to separate the A and B addresses in a JA or JNA order.

The B address of a JC or JNC order may be written as five sexadecimal characters or the equivalent decimal number. If a symbolic name is used, then it is used in the normal way and sets only the primary address and the index bits must be specified in the normal way.

The C address of an input-output order can be written in a special way. This address is used to specify the type of input-output (card, tape, drum or flexowriter) and other information such as tape unit, drum channel, read or write, etc. Appendix C lists the symbolic names that can be used to specify the C address on input-output orders. (These special symbols must be in the third address of the order and not in the first or second address.) The order type used (CARD; TAPE; DRUM; FLEX) also sets the C address (the last sexadecimal) for reading (except FLEX sets for writing) from that equipment. If the order type specifies the equipment, then the C address need specify only the additional information that is needed and R(or READ) or W(or WRITE) may be used to specify the direction of information flow for that equipment. (If equipment is specified both places, the equipment specified in the C address will be used.) An order type of ZERO sets the C address to 0807 for clearing the memory. (CLEARM may also be used in the C address for the same purpose.) The entire 20 bit C address may also be specified by one sexadecimal address.

The addresses in the SI, SIJ or SETJ orders may be written in the standard way or in the same manner as followed in the SET statement. Therefore SI(I=3)J=0 % is the same order as SI(3,I)0,J %. The addresses in the II, IIJ or INCJ orders may be written in the standard way or in the same manner as allowed in the INC statement. Therefore II(I=I+1) J = J + 4)% is the same order as II(1,I)(4,J)%. Note that these assembly orders must not have more than three addresses.

A decimal number may be written in place of an address by preceding the number with an *. Any type of number allowed in DEC may be written. A sexadecimal constant may be written in place of an address by preceding it with a / or **. Any type of constant allowed in SEXA may be written.

Addresses that are either omitted from the end of an assembly order or are left blank by the use of parentheses are set to 00000. Each address should be followed by) and the following (is optional.

Some examples of BRLESC assembly orders are:

LOC.	O.T.
	FA (A)B+3)T1% FMA(X,I)0)F3,J+2%
	TPS(3.2)MASK)5.7)%AX(Q)*13)Q%
	SH(R3)BTCR20)R3% C-X(S1)*0)8.4%
	CB8(W,I-2)/oLZ)SELF-4%% COMMENT
	BRLESC % TAPE ())WFMR-3A%

The following instructions have been added to BRLESC and the FORAST compiler to facilitate the addressing of a large memory. It is not necessary to use these instructions in writing normal FORAST programs as the compiler automatically inserts the necessary instructions required for large addresses. However, in subroutines and some FORAST programs, it will be necessary to use some of the following instructions:

SEXA.	SYMBOLIC	COMPUTER DESCRIPTION
11	SIL	$a\alpha$(20 bits) → (b) and $c\gamma$ → (β)(index address.)
12	IIL	$a\alpha$ + (b) → (b) and $c\gamma$ + (β) → (β).
14	JL	Jump to $c\gamma$ (20 bit address).
15	JSL	Jump to $c\gamma$. Also A → (2); B → (3) and (NI) →(1)
05	JS	Same as JSL except jump to C instead of $c\gamma$.
160	I	Leading 4 bits of a are zero, the next 12 bits are the index address for the first address of the next order that is done as specified in $c\gamma$, the next 12

SEXA.	SYMBOLIC	COMPUTER DESCRIPTION
		bits are the index address for the second address of the next order and the next 12 bits are the index address for the third address of the next order. cγ is a 20 bit address that is the address of the next order and that order uses its aα, bβ, and cγ as 20 bit addresses. Control does not stay with the order specified in cγ; after it is performed, control returns to the next order after the I order.
168	INI	Same as the I order except NI (next instruction counter) is advanced when the order specified in cγ is done. This allows cγ to specify the very next order following this INI order. (If cγ is not SELF+1, the next order is skipped.)
17	LPIL	Same as LPI except the leading bit of β is spread to the left to form a 20 bit address when the effective address B is computed.
19	SLJL	aα →(b) and jump to cγ + (β) where β must be an index address.
1K	ILJL	aα + (b) →(b) and jump to cγ + (β) where β must be an index address.
1S	EAL	aα + (b) → (β) where β is an index address.
1F	IM or MI	Integer Multiply where only the last 5 sexadecimal characters of β are used as the mutiplier and the integer product is stored in C.
1L	RCL	Read the real-time clock into A. B must be 1 and jump to C. (See R.CLK subroutine.)
LX	IT	A →(4); B →(5); C →(6) and jump to 040.

SPECIAL COMMENTS ON USING THE ABOVE SYMBOLIC ORDER TYPES IN FORAST

SIL is translated much like a SET statement where the general form of (Index address = address to put into the index) is the preferred way of writing these instructions. Any number of indexes may be set and a GOTO may be used after the last one to get a SIJL order for the last one. (Note that only two indexes may be set in each computer word when SIL is used.)

Examples: SIL(I = A)J = B)C = 0)KK = 19462 %

SIL(A,I)T = 3)GOTO(BOX 3)%

Note: A,I is the same as I = A in a SIL statement.

IIL assumes addresses in the same general form as the INC statement. (I = I + 3) is an example of the preferred way of writing these instructions. Any number of indexes may be increased (or decreased) and it may be terminated with a GOTO that causes a IIJL order to be used for the last one.

Examples: IIL(I = I + 1)J = J - 3)%

IIL(K = K - 4200)T = T + 0800)V = V + 3)GOTO(7.2%

Note: 1,I is the same as I = I + 1.

SIJL and IIJL must have only <u>one</u> index that is to be set or increased and a jump address.

EAL must have only <u>one</u> effective address to be computed and may optionally be followed with a normal cy address that is not directly used by the computer. If there isn't an = character after the first name in the first address, the instruction is processed in FORAST as a normal three address instruction.

Examples: EAL(I = A,J)%

EAL(K = B,M - 2)Q,I %

EAL(A)I,J)% Is same as first example .

Four addresses may be written in the I and INI orders, three index addresses and a non-indexable fourth address that is the location of the next order to be done. (See above description of the I and INI orders.) If the fourth address is omitted, SELF + 1 is used and the order type will be set to 168 even when I was the symbolic order type. The index addresses can not contain increments. If an increment is desired, it must be written with the primary addresses in the order that is referenced by the cy of the I or INI order. The order referenced by an I or INI order should be preceded by "LA/" if any addresses written in that order might be assigned to an address longer than 14 bits. The LA/prevents the compiler from automatically inserting another I order.

Examples: I(I)J)K)A + 4 %

INI(M))V % LA/TP(A + 1)/LL)R %

XV. OPERATION AND SPEED OF THE ORDVAC TRANSLATOR.

The ORDVAC translator program is now on magnetic tape. The program to be translated must be preceded by a "Tape Start" card and the data to be read by the translated program should be placed immediately after the END card. A blank card must be put at the end of the deck; otherwise, the last card will not be read.

The ORDVAC translation is done in the following sequence:

1. The symbolic program deck is read at maximum speed (200 cpm) and concurrently changed to alphabetic computer words of five characters each. The non-blank words are saved on the drum until the entire program has been read.

2. The first part of the translation process calls the alphabetic words from the drum and generates machine orders and stores the partially translated code back on the drum. It creates a dictionary and constant pool in the memory.

3. After the END card is translated, the automatic assignment of all unassigned symbols is done.

4. The memory is cleared to zeros (except for 0-0L; 040-end of constant pool; and about ONOO - OLLL).

5. The partially translated code is called from the drum, the translation is completed and the code is stored directly in the memory where it belongs for running. (Programs should not be stored below ONOO in the memory).

6. All of the subroutines are read and the ones used by the program are stored in the memory. They are stored backwards from OJ70 unless the program has specified that they be stored elsewhere. The input-output routine is always stored at OJJO. The subroutines are on relocatable binary cards and therefore do not require translation.

The time required for translating a FORAST program consisting of C cards on ORDVAC can be roughly approximated by the following formula:

$$\text{time in sec.} = 20 + C$$

The 20 seconds in the formula is the time required to read the translator program from tape. The time to actually read and translate a symbolic program card is almost a secord in an "average" size program of about 150 cards with about two statements per card. (This time will vary considerably with the length and number of statements on each card, the number of different symbols used, the length of the program, etc.)

XVI. OPERATION AND SPEED OF THE BRLESC TRANSLATOR.

The BRLESC translator is on magnetic tape and is designed for use with the "tape start" button. To translate a program, it should be placed in the card reader and the data that it reads should immediately follow the END card. The tape start button is then pushed and this causes the program to be translated and run. (Tape switch 15

must be set to the tape unit that has the compiler tape on it and tape switch 7 must be set to a tape unit that contains a "temporary" tape. Manual read switches 36 and 35 should be down for card input and output. The compiler and the translated program will read magnetic tape using tape switch 6 if manual read switch 36 is up and they will put all standard output on tape using tape switch 8 if manual read switch 35 is up. For tape input or output, 80 characters on tape are the same as one card.)

The BRLESC translation is done in the following sequence:

1. The symbolic program is read at 800 cards per minute and most of the translation is done concurrently with the card (or tape) reading. The partially translated code is put in the memory and on a temporary tape if the memory gets full. A constant pool and dictionary is created and kept in the core memory.

2. After the END card is translated, the automatic assignment of all unassigned symbols is done.

3. The memory is cleared to zeros. (0K-03F and end of constant pool to about 0N00).

4. The partially translated code is called from tape, the translation is completed and the code is stored directly in the memory where it belongs for running. (Programs should not be stored in the last 8000 words of memory.)

5. All of the subroutines are read from the compiler tape and the ones used by the program are stored in the memory. They are stored backwards from 0SK0 unless the program has specified that they be stored elsewhere. The input-output routine is always stored at 0N80 and the N.PROB (next problem) routine is always stored at 0N70-0N7L. The subroutines are in a binary re-locatable form and therefore do not require translation.

The BRLESC FORAST compiler is one of the fastest compilers that is in use at this time. It requires about two seconds of tape time for itself and easily translates as fast as it can read the symbolic program cards at 800 cards per minute, regardless of how many statements are on each card. The time required for translating a program consisting of C cards on BRLESC can be approximated by the following formula:

$$\text{time in secs.} = 2 + C/13 + C/100$$

The 2 seconds is compiler tape time, the C/13 is card read time and C/100 allows time for reading the "temporary" tape and completing the translation. If the symbolic program is put on tape, then the translation will be about 3 times faster than from cards. (The time required to translate from tape will vary considerably with the program to be translated. For most programs, it will vary between 2 to 5 times faster than the speed of card translation.)

Since the BRLESC translator is so fast, it is not necessary for programmers to obtain binary card decks for production running. It would take about as long (possibly longer) to read the binary deck as it takes to translate the symbolic deck.

XVII. INSTRUCTIONS FOR USING FORAST TRANSLATORS TO RUN PROGRAMS.

A. ORDVAC

A "tape start" card must be placed in front of the symbolic FORAST program that is to be translated and the data deck for the program is to be placed after the END card and should have the usual blank card at the end of the entire deck. (It is usually permissible to also have a blank card between the translator and the first data card since READ statements ignore the first card when it is blank.) This deck is then run in the standard way.

If a teletype print occurs before compilation is completed, this indicates that the program contains an error (e.g. a mispunched card) and will not compile. In this event, a dictionary is always punched on cards so that the programmer might also check it for indications of other errors.

The ORDVAC compiler halts at 009 after finding errors except for a few types of errors for which it may halt elsewhere. If the ORDVAC should halt during the translation without a teletype print, then the contents of 04JJ should usually help the programmer to find the trouble. (04JJ will contain the card counter.) Do not toggle past any halts that occur during compilation.

After the compilation is completed, the program begins to run without any halt in between. If the running program halts at any place other than OLN7 (counter has OLN8) or OL9L (counter has OLKO), a manual jump to 006 should be done unless 006 contains a jump to OLN7 anyway. (OLN7 is the standard N.PROB halt and OL9L is trying to read more cards.) Any teletype output should be saved for the programmer.

XVIII. INSTRUCTIONS FOR RUNNING FORAST PROGRAMS ON BRLESC.

To compile and run a FORAST program on BRLESC, tape switch 15 must be set to the tape unit that holds the compiler tape and tape switch 7 must be set to a unit that the compiler may use for temporary storage. Manual read switches 36 and 35 are used to control the type of input and output. Switch 36 must be down if the input (program and data) is to be read from cards and it must be up if the input is on tape. (Tape switch 6 must be set to the unit holding the input tape.) Manual read switch 35 must be down if the output is to be punched on cards and it must be up if the output is to be put on tape. (Tape switch 8 controls the unit used for the output tape.)

When the input is on cards, the program deck and its data are loaded into the card reader and the tape start button is pushed to begin the compilation of the program.

The tape start button is always used to start compilation. It is possible (but improbable) that this will not properly begin compilation if the compiler tape (15) is left on certain blocks. If after a tape start, compilation does not proceed as expected, just do another tape start and compilation should begin. The compiler tape is generally self-correcting with regard to tape start, tape errors and end of tape. It contains many copies of the compiler with a file mark between copies. Almost all of the blocks move to the beginning of the next compiler if a tape start is done when the tape is not at the beginning of a compiler copy. All blocks are read and parity checked (by programming) before they are used and if an error is found, the block is reread once and if the error persists, the tape is moved to the next compiler copy and the block is read from the copy and checked, etc. To save time and avoid wear at only the beginning of the tape, successive problems will use successive copies of the compiler. If the compiler tape comes to the end of all the copies on the tape, it moves itself back twenty copies and re-uses them.

After a problem is compiled, it automatically begins to run unless the compiler discovered one or more errors in the program. If an error is found, during compilation, a card is punched (or tape is written if using tape output) for each error, the "dictionary" is printed, and the computer halts at ON40 with a B address of OLLL in the halt order. Initiate past this ON40 halt only if the problems have been "stacked" as described below.

If the problem comes to an abrupt or unusual halt or if it cycles, note this fact along with the NI and PO registers for the programmer's information and then do a jump to 058. Do not jump to 058 if the halt was at ON40 which is the standard halt when a problem is completed and is ready to begin the next problem. After going to 058, the computer should soon halt at ON40. (Going to 058 allows the programmer to do some extra printing after his problem runs into trouble. It also provides for checking and re-setting the overflow indicators and it causes the time "charge" card to be punched.)

To stack problems for tape input, one (or more) PROB cards must be placed at the end of each problem deck. The problem decks are put onto tape by the off-line converter or by a program on the 1401. Initially the tape must be set up for writing by pushing the "prepare to record" button; then each problem is put into the card reader separately with a PROB card at the end and is put onto tape. The block length must not be more than eleven cards per block. (Each problem may have a different block length, but it is best to actually use eleven card blocks as it saves time over using shorter blocks.) The "record remains" button and then the "write intermediate file mark" button must be pushed at the end of each and every problem. When all of the problems have been put on tape, an extra "END TAPE" problem should be put on the tape. Then the "record remains" and then the "write final file mark" and then the "rewind" buttons should be pushed. (The "END TAPE" problem put at the end of the tape takes care of rewinding both input and output tapes when manual read switches 36 and 35 are up and comes to an absolute halt at ON50.)

When running programs that have been stacked on tape, it is permissible to do a tape start at any point on the tape and the next problem on the tape will be done. However, if the output of a problem is being put on tape instead of cards, then a tape start before an ON40 halt may cause the problem that just finished to not print its last few "cards" of output on tape. Thus when tape output is used, the operator should always attempt to get to the ON40 halt by jumping to 058 after any other halts. At an ON40 halt, it is better to initiate than to do a tape start. The α address of the halt order at ON40 (and of 077) contains an integer number (1,2,3,...) that shows how many problems have been run since the last tape start.

The leading part of 07J usually contains the actual problem number assigned to the problem (C-916, 0-780, etc) in six bit characters while the problem is running. (Only rightmost 60 bits are used.)

If the nth problem on an input tape should be run without running the preceding problems, this can be done by moving the tape (n-1) file marks before doing a tape start. (There is a file mark between each problem on the tape as well as one at the beginning. The tape must be stopped ahead of the file mark that precedes the next problem that is to be done.) It is possible to skip n problems on the tape by moving the tape forward n file marks.

When tape output is used, the tape contains variable length line control characters for off-line printing. There is a file mark between each problem so that the printer may be stopped at the end of each problem and n problems may be skipped by moving the tape n file marks. (Move n+1 file marks to skip the first n problems because of the extra file mark at the beginning of the tape.) The last problem on the tape will print only two lines and the last line will say "LAST PROB END TAPE".

When card output is used, the output of each problem should be taken out of the hopper and kept separate from the other problems. If a problem has halted at ON40, it is not necessary to read down one blank card as the program has already done this. It is possible that some binary cards will be punched when using tape output.

When problems are stacked on an input tape, a list of the problems that are on the tape should be made. Along with the problem number and programmer's name, the maximum run time and any special instructions should be noted so that the computer operator may have this information available. This list should be kept with the output tape so that people know what is on it. Each output tape should be kept for at least one week.

SUMMARY OF INSTRUCTION FOR RUNNING FORAST PROBLEMS ON BRLESC:

1. Tape control:
   a. Compiler on tape switch 15, is activated by a "tape start".
   b. Tape switch 7 used for temporary tape.

c. For tape input using switch 6, set manual read switch 36 up.

d. For tape output using switch 8, set manual read switch 35 up.

2. Halts:

a. N40: end of problem, initiate if problems are stacked. (α is the no. of problems done since the last tape start; also in α of 077. If β = LLL, there was a program error detected by the compiler.)

b. N50; end of all the problems on an input tape 6.

c. All other halts or cycles; note PO and NI registers and then do jump to 058. It should soon get to N40. If it doesn't, then must use card deck to get "charge card" and then do a "tape start" to begin next problem.

3. Stacking problems on tape (done on off-line unit):

a. Get tape at its beginning and push "prepare to record" button.

b. Set block length to eleven (11) or less.

c. Set switches for card-to-tape operation.

d. Put 1 (or more) PROB card at end of problem.

e. Initiate to put problem on tape.

f. Put problem number, programmer's name, maximum run time, and any special instructions on a list.

g. Push "record remains" button.

h. Push "write intermediate file mark" button.

i. Repeat steps d through h for each problem.

j. Put special "END TAPE" problem on tape. (Don't need PROB card.)

k. Push "record remains".

l. Push "write final file mark" button.

m. Push "Rewind" button.

4. Stacking problems on tape can also be done by IBM 1401 computer by using the proper program.

5. Listing output tape or producing cards off-line:
   a. Set for tape-to-printer or tape-to-card operation.
   b. Set to stop at each file mark. (Stops between each problem.)
   c. Do variable length line list unless special instructions say otherwise.
   d. Start a new page at beginning of listing each problem. (Is done automatically unless the next problem requires 80 or 160 column list.)
   e. The last problem on the tape prints just 3 lines of print, the last line says "ND TAPE".
   f. To skip n problems, skip n file marks. (Skip n+1 file marks if starting at beginning of tape.)

XIX. MISCELLANEOUS COMMENTS.

1. The method for arithmetic formula translation is a modified and expanded version of that described by J. H. Wegstein in [3].
2. The end of statement symbol "%" is not required at the end of the last statement on a card. The end of a card automatically causes the end of a statement except when the next card has "CONT" in cols. 7-10.
3. There is no limit to the number of CONT cards permitted.
4. Two adjacent symbols "%%" causes the rest of a card to be ignored so that it may be used for comments.
5. A statement of "ORDVAC %" causes BRLESC to ignore the rest of the card (and any successive CONT cards).
6. A statement of "BRLESC %" causes ORDVAC to ignore the rest of the card (and any successive CONT cards).
7. Any right parenthesis immediately preceding % may be omitted. Wherever parentheses are used only to separate names or numbers, a single right parenthesis is necessary and sufficient. The following left parenthesis is optional.

8. Blank cards in the program are ignored. Blank columns are ignored except in a "string of characters" in a PRINT or PUNCH statement and within a symbolic ORDVAC order type in cols. 7-10.

9. The translators do not check the memory space required by a program against the space available. This may be partially checked on ORDVAC by making sure that the "% SUBS." address is the largest in the dictionary (except for N.PROB). The next largest address will usually be the last one that is marked with an M. However this doesn't necessarily check the total amount of storage required by the running program. ORDVAC at present has about 3200 words available for a program. On BRLESC, programs have a minimum of about 30,000 words available and each program should leave at least 624 words at the end of the memory for tape output buffers. The program may be further restricted by 4000 words or more if the compiler stored "large address" orders in the next to last segment of 4096 word segments. (The dictionary print includes a statement of where "large address" orders are stored.)

10. The best way of code checking most problems is to insert extra PRINT statements. They can be used to check both the path of the program as well as the computed data. Be sure that the printed results can be identified with the proper PRINT statement. It is a good idea to include some alphabetic information with each print for this purpose. This method of code checking is practical because of the fast compilation. The M.DUMP idea (See XIII) may also be used to do some printing after a program runs into trouble.

11. It is recommended that the PROB, BLOC, SYN and LAST cards be at the beginning of the program and in that order. All SYN cards must be ahead of the program on ORDVAC and BLOC must precede the usage of any of the block names it defines.

12. The physically last card of every program must be an END card.
13. Location names appearing in LOC, DEC, SEXA, DEC =, and FORM are assigned storage space at the point they appear in the program. These types of cards must not be inserted in the midst of a program. They should not follow any executable statement except a GOTO statement and should normally be at the beginning or end of the program.
14. Integer numbers must not be used as symbolic locations. They will be used as decimal absolute addresses. Numbers with decimal points may be used as symbolic locations.
15. There are no restrictions on transferring into or out of the range of COUNT statements (except for the use of COUNT without the "IN clause" on BRLESC.) All index registers always hold their current value. However since BRLESC has only 63 index registers, a FORAST program on BRLESC must not use more than 54 index names unless some are assigned the same storage by a SYN card. Any memory cell on ORDVAC may be used as an index register.
16. Indexing must correspond to memory allocation because the value in an index register is <u>added</u> to the primary address to get the effective address. Thus if X1,X2, ... are stored in every other memory cell and addressed by writing X1,I; then I must be increased by <u>two</u> to access consecutive X's. Note also that X,I when I = 1 is not necessarily the same as X1,I when I = 0 unless X is defined as part of a "BLOC" that is assigned consecutive cells. To index through a "BLOC", the primary address must be a member of that block. In these respects FORAST uses more of the philosophy of an assembler rather than that of a compiler.

17. BRLESC index registers are not full words. Therefore it is not permissible to store floating point numbers in cells that are assigned to index registers. Integers larger than 16,383 should not be stored in them either. Care must be taken when using negative numbers in index registers as the sign bit will always be zero when they are used as full words in the arithmetic unit.

18. It is easier to define a "universal" language than it is to use that language to write "universal" programs. Below is a list of some of the differences between ORDVAC and BRLESC that may cause the same program to do different things on the two machines. These differences may cause the programmer to want to use slightly different programs on the two computers or to run a program only on one of the computers.

	ORDVAC	BRLESC
Compute Speed:	1	at least 20 times ORDVAC
Input-Output	Cards only:	magnetic tape
	200 cpm in.	cards in: 800 cpm
	100 cpm out.	cards out: 250 cpm
Word length:	40 bits	68 bits
	$\pm$38	$\pm$155
Fl.Pt.No. Range:	10	10
Fl.Pt. Rounding:	no	yes
Index Registers:	4096 full words	63 (20 bits)
Memory Size:	4096	36,864 or 53,248
Instructions per word:	2	1
Address length:	12 bits	16 bits

19. The information on DEC, DEC =, SEXA and FORM cards may be preceded by "ORDVAC (" or "BRLESC (" to allow the information on that card to be stored only on the machine memtioned. The other computer will ignore the rest of the card (and any successive CONT cards). There must not be any symbol in the statement field to the left of the computer name.

20. SET, INC, COUNT, and SETEA statements should not use index registers that have just been set or changed in the same statement. BRLESC doesn't always store the previous index result before accessing the next one within the same BRLESC instruction.

21. CLEAR and MOVE statements may use one to three index registers. On ORDVAC, the first three cells of formula temporary storage are used. On BRLESC, index registers 2,3, and 4 are used.

22. It is not permissible to READ and PRINT zero numbers (by using "(I)NOS.AT" where I = 0) or to CLEAR or MOVE zero numbers. A max count of zero in a COUNT statement is the same as a count of one.

23. An IF statement always compares the contents of cells. While it may seem that statements of SET (A = A2 % and IF-INT(A=A2)GOTO(B)% should cause a transfer to B, they will not. The IF statement checks the contents of A against the contents of A2 rather than the contents of A against the address A2.

24. The translators produce efficient but not necessarily the best possible code. Each operation is generally coded in the best way with respect to the previous operation. The translators make only one pass, do not re-arrange expressions nor waste time looking for common sub-expressions. Common sub-expressions, expecially those involving the use of subroutines, should be evaluated in separate statements and re-named by the programmer. The efficiency of the

translated code is almost entirely dependent on the efficiency and skill of the programmer. In any translator, the inefficiencies that can be removed by the translator are always a very small percentage of the inefficiencies that a programmer can put into a program. It is worthy of comment that FORAST does not allow some excess generalities that tend to produce inefficient code and protracted tape searches. They are (1) mixed integer and fl. pt. arithmetic expressions, (2) variable multi-dimensional indexing, (3) arithmetic expressions everywhere, (4) ALGOL type "procedures", (5) dynamic storage allocation and (6) optimization of index register usage. (Optimization of index registers is not necessary because of the fact that both machines have 63 or more index registers.)

ACKNOWLEDGEMENTS

Mr. Alfred Anderson programmed the decimal input-output routines for both computers and the modified-Adams subroutine. Mr. Donald Taylor programmed the matrix subroutines, the Runge-Kutta-Gill subroutine and Simpson's integration subroutine. Miss Helen Mark programmed the random number subroutines NRNOS1 and NRNOS2 and Mr. Barry Rodin programmed NRNOS. for BRLESC. Miss Viola Woodward programmed the Cowell and Bessel subroutines. Acknowledgement is made to Mr. Peter Smith for his editorial assistance.

LLOYD W. CAMPBELL GLENN A. BECK

REFERENCES

1. LESER, T. and ROMANELLI, M. Programming and Coding for ORDVAC, BRL Report No. 997 (October 1956).

2. CAMPBELL, L. and BECK, G. The Instruction Code for the BRL Electronic Scientific Computer (BRLESC), BRL Report No. 1379 (November 1961).

3. WEGSTEIN, J. From Formulas to Computer Oriented Language, Communications of the ACM, March 1959.

APPENDIX A. SYMBOLIC ORDVAC ORDER TYPES

OLD SYMBOL	FORAST SYMBOL	OLD SYMBOL	FORAST SYMBOL
+	+	E	E
-	-	E'	E'
(+)	(+) or + H	oE	oE
(-)	(-) or - H	oE'	oE'
|+|	1 + 1	M	M
|-|	1 - 1	1M	1M
[+]	+ H V	oM	oM or OM
[-]	- H V	⊕	OR
A+	A+	(R)	(R)
A-	A-	C	C or C+
A(+)	A(+) or A + H	C'	C' or C'+
A(-)	A(-) or A - H	U	U or GOTO
A|+|	A1+1 or A + V	U'	U'
A|-|	A1-1 or A - V	oU'	oU'
R	R	ZX	ZX
÷	/	ZU	ZU or HALT
X	X or *	T	T
XU	XU or *U	P	P
(X)	(X)	IBM IN	IBMI
←	LS	IBM OUT	IBMO
←0	LSo	Drum read	DR or DRI
→	RS	Drum write	DW or DRO
→1	1RS	"A"	'A'
←8	SS	U*	U*
←18	1SS	Index	I
←9T	SST	Mag.Tape Select	TS
		Mag.Tape Transfer	TT
		Mag.Tape Parity Check	TP

APPENDIX B. SYMBOLIC BRLESC ORDER TYPES

SEXA.	SYMBOLIC O.T.
2	A or +
3	S or -
4	M or *
5	D or /
6	C or C-(C+ is 62)
7	SQRT or SQ
8	SH
9	TP
K	B
S	CB
N	CNB or CN
J	PMA
L	IT
00	HALT
01	SET or SI
02	INC or II
03	LP or LOOP
04	J or JUMP
05	JS
06	J+
07	J-
08	IO or CARD or TAPE or DRUM or TYPE or ZERO
09	SIJ or SETJ
0K	IIJ or INCJ
0S	EA
0N	JA
0J	JC
0F	NOP
0L	RSW
10	MMF
11	SIL
12	IIL
13	LPI
14	JL
15	JSL
160	I
168	INI
17	LPIL
18	MMB
19	SIJL
1K	IIJL
1S	EAL
1N	JNA
1J	JNC
1F	IM or MI
1L	RCL

SYMBOLIC PARAMETERS

F	Floating Point
X	Fixed Point
A or +	Accumulate Result
V	Absolute Value of Operands
U	Unnormalized Floating Point
N	Normalized Floating Point
R	Use R Register and Fixed Point

SYMBOLIC SHIFT CODE

D	Double Length
B	Boolean
T	Tags Included
R	Round (if precedes direction)
C	Cyclic
Z	Zero Last 8 Bits.
L or +	Left
R or -	Right

Magnitude of shift may be either decimal or sexadecimal but must be written last.

BOOLEAN FUNCTIONS

0	0	8	xy (And)
1	$\bar{x}\,\bar{y}$	9	$xy + \bar{x}\,\bar{y}$
2	$\bar{x}\,\bar{y}$	10(K)	y
3	$\bar{x}$(not)	11(S)	$\bar{x} + y$
4	$x\,\bar{y}$	12(N)	x
5	$\bar{y}$	13(J)	$x + \bar{y}$
6	$x\,\bar{y} + \bar{x}\,y$ (Ex.or)	14(F)	$x + y$ (In.or)
7	$\bar{x} + \bar{y}$	15(L)	1

APPENDIX C. SYMBOLIC C ADDRESS FOR BRLESC INPUT-OUTPUT ORDERS

CARD	DRUM		
READ 68	DR - t - s	where	t = track no.
			s = sector within track
READ 80	DW - t - s		and both t and s must be
			decimal or sexadecimal.
PUNCH 68	CLEARD - t - s		CLEARD means to store zeros
			on the drum.
PUNCH 80	CLEARM		CLEARM may be used to store
			zeros in the memory.

TAPE

All of the following tape commands must be followed by a dash (minus sign) and the tape unit number in decimal. A trunk may be specified by following the unit number with A or B. If no trunk is specified, tape read orders will use trunk A and tape write orders will use trunk B. A "-P" should also be used on "ten character" read orders if the computer should store a sign bit when a tape error occurs. The 10 or 12 that follows R, RN, and W below indicates the ten character per word mode or the twelve character per word mode of reading or writing magnetic tape. See [2] for a description of the BRLESC magnetic tape system and the meaning of the following symbols.

READ		WRITE
R	PR	W
R10	REW	W10
R12	REWIND	W12
RN10	REWI	PW
RN12	UNWIND	WFM
MF	UNW	WFMR
MOVEFO		BW
MB		GAP
MOVEBA		
MFMF		
MFMB		

APPENDIX D. NUMBER OF ELEMENTS IN TRIANGULAR ARRAYS

n	not augmented	augmented	n	not augmented	augmented
1	1	2	51	1326	1377
2	3	5	52	1378	1430
3	6	9	53	1431	1484
4	10	14	54	1485	1539
5	15	20	55	1540	1595
6	21	27	56	1596	1652
7	28	35	57	1653	1710
8	36	44	58	1711	1769
9	45	54	59	1770	1829
10	55	65	60	1830	1890
11	66	77	61	1891	1952
12	78	90	62	1953	2015
13	91	104	63	2016	2079
14	105	119	64	2080	2144
15	120	135	65	2145	2210
16	136	152	66	2211	2277
17	153	170	67	2278	2345
18	171	189	68	2346	2414
19	190	209	69	2415	2484
20	210	230	70	2485	2555
21	231	252	71	2556	2627
22	253	275	72	2628	2700
23	276	299	73	2701	2774
24	300	324	74	2775	2849
25	325	350	75	2850	2925
26	351	377	76	2926	3002
27	378	405	77	3003	3080
28	406	434	78	3081	3159
29	435	464	79	3160	3239
30	465	495	80	3240	3320
31	496	527	81	3321	3402
32	528	560	82	3403	3485
33	561	594	83	3486	3569
34	595	629	84	3570	3654
35	630	665	85	3655	3740
36	666	702	86	3741	3827
37	703	740	87	3828	3915
38	741	779	88	3916	4004
39	780	819	89	4005	4094
40	820	860	90	4095	4185
41	861	902	91	4186	4277
42	903	945	92	4278	4370
43	946	989	93	4371	4464
44	990	1034	94	4465	4559
45	1035	1080	95	4560	4655
46	1081	1127	96	4656	4752
47	1128	1175	97	4753	4850
48	1176	1224	98	4851	4949
49	1225	1274	99	4950	5049
50	1275	1325	100	5050	5150

CODING FORM

PROBLEM Summary of Pseudo Order Types

CODER DATE PAGE

LOCATION 1–6	ORDER TYPE 7–10	FORMULA STATEMENTS A 11–20	B 21–30	C 31–40	COMMENTS 41–80
	PROB	W-622 J.R. BROWN JULY 1962 SATELLITE DATA FITTING			
---	DATE	AUG.20,62			Specify current date.
---	COMM	Entire card is comment.			
---	MODE	FLT or FIX or FU			Integer mode not allowed.
---	BLOC	(A - A24)(0200/Y17 - Y41/3) M1 - M4/I %			Define blocks.
---	SYN	(A2 = XDOT) Q4 = 3A8 = X')(N = 9) %			Assign same address to different names.
---	CONT	Used to continue from previous card.			
---	LAST	(X) Y) Z %			These names are assigned last.
A	LOC	B)(C) R %			Used to get names assigned in desired sequence.
	DEC	(16.1) F14) X.214 B-1) I22) %			Store decimal constants.
	DEC =	TBAR = 14)(EPS = .1-5) PI = F3.14159) J6 = I400 %			Name and store dec. constants.
	HEXA	(3K) 49Z) 64 ZFKS %			Store sexadecimal constants. Z indicates one string of zeros.
	FORM	(9 - 10) 1 - 3) 4 - 5) 6 - 3 - 5 %			Store format for decimal input or output.
---	LIST	(S.CODE) B.CODE %			Used to get dictionary and/or code printed.
	STOR	31.4 + 3 %			Is same as if were a location on the next card.
---	NOS.	OLO %			Used to change address of constant pool.
---	FITS	080 %			Used to change temporary storage used within formulas.
---	ASGN	(Q = 8) %			Used to re-assign symbolic names.
---	SUBR	(ARCTAN = 0800) FIX %			May be used to assign subroutine storage or define new functions.
	ALFN	Used to store alphanumeric constants.			
	O.T.	Used to define new pseudo BRLESC order types.			
	END	GOTO(START)%			Must be last card of program.
		Dashes in the location field above indicates that this field is not used on ORDVAC.			

ORDBG Form 2534-(R), 2 Feb 60 Appendix F. PROBLEM Summary of Assembly Code and Statements

CODING FORM CODER DATE PAGE

LOCATION 1 6	ORDER TYPE 7 10	FORMULA STATEMENTS 11 A 20 21 B 30 31 C 40	41 COMMENTS 80
		ORDVAC Assembly Orders	
	F(+)	B,I	
	FM	T1 % Any Comment allowed here after %.	
	MACH	+(A,J)% M (B,I+1)% F + (R1) % F/(S4)% A + % FM(X3) % U(6.2)%	
		BRLESC Assembly Orders	
	FA	X) Y) Z,I3 % JS(Z,I3) R) SIN % B12(X))Y %	
		SHX(T1 + 2)(BTCR14) 0 % TAPE ( )) PW - 4B %	
		Examples of Arithmetic Formulas.	
		Y = 0 % A = B % R = S = T = 0 % Y,I = A + B/C - EXP(Q + VI)%	
		S = R ** 3 % V = - (V - VS * 6.417)R % 2C4 = (-A4) B6 * C,(I-1) - W,K %	
		FLT(R,(I + 2) = S,I(X ** 2 + Y ** 2)% FIX(T2 = .3(A - R)% INT(I1 = I - J)%	
		Examples of English word statements allowed.	
		GOTO(LOC 3)% GOTO(8.3)% GOTO(,E)% GOTO(N.PROB)%	
		SET(I = 0) J = 1) E = E2)% SET(N = 4) K = N + 1) GOTO(6.4)%	
		SETEA(I = A,J) K = R,T - 3) P = X,V1 %	
		INC(I = I + 1) J = J - 2 % INC(R = R + 084) GOTO (19.2) %	
		COUNT(10)IN(I)GO BACK TO(39.1)% COUNT(N + 2)BY(2)IN(J)GOTO(,B)%	
		IF(X)IS + OR(Y * B = 0)GOTO(BOX 3)% IF(X > A + B < 14)AND(V = 0)GOTO(3.)%	
		IF - INT(I + J > = 5)OR(A,I = B,J)WITHIN(.001)GOTO(S3)%	
		CLEAR(15)NOS.AT(A)% CLEAR(N/3)NOS.AT(B/6)%	
		MOVE(246)NOS.FROM(X)TO(Y)% MOVE(J + 14)NOS.FROM(V,K/-1)TO(A)	
		READ(X1)X2)X3 % READ(5)NOS.AT(M/2)% READ-FORMAT(F6/4)-(16)NOS.AT(X1) %	
		PRINT(X)FX)RX % PUNCH(M + 2)NOS.AT(S,K3 + 2/4)X % PRINT < X TOO BIG > (X) %	
		ENTER(SINCOS)X)SIN X)COS X % ENTER(MAT.MP)A1,1)(B1,1)C1,1)3),I)5 %	
		HALT % HALT(3)%	

LOCATION 1–6	ORDER TYPE 7–10	FORMULA STATEMENTS 11–80 (A 11–20 | B 21–30 | C 31–40 | COMMENTS 41–80)
		Statement of Problem: Read in sets of X,Y and Xi for i = 1,2,...,8. Compute s = 3XY + $\sum_{i=1}^{8}$ Xi
		and print X,Y and S. A card with an X value greater than 10^{35} will follow the
		last set of data. All nos. are floating point numbers using standard 12 column fields.
	PROB	C-905 L.W.C. AUGUST 1962 SAMPLE PROBLEM
	BLOC	(X1 - X8)
1.1		READ (X)Y)8)NOS.AT(X1)$ IF(X > S1)GOTO(N.PROB)$
		SET(I = 0)$ S = 3(X * Y)
2.1		S = S + X1,I
		COUNT(8)IN(I)GOTO(2.1)$
		PRINT(X)Y)S $ GOTO(1.1)$
S1	DEC	(1. + 35)
	END	GOTO(1.1)$
		Statement of Problem: Read an n by n square matrix, find and print its inverse. Read another
		matrix, etc. Assume standard input with a blank card at the end of each matrix. Print the
		inverse with five 15 col. fl. pt. fields and a 5 col. card counter that starts at 1 for each
		matrix. Also print a blank card between each matrix done.
	PROB	C-906 L.W.C. AUGUST 1962 MATRIX INVERSION
	LAST	A1
START		READ(16383)NOS.AT(A1)$ ENTER(CVITOF)9)N $ N = SQRT(N)$
		ENTER(CVFTOI)N)NI $ ENTER(MAT.INV)A1),NI $
		PRINT-FORMAT(F)-(,9)NOS.AT(A1)$
		ENTER(PRINT BLANK)$ ENTER(ZEROCC)$ GOTO(START$
F	FORM	(9 - 15) 1 - 5) 5 - 1 - 5 $
	END	GOTO(START)$

ORDBG Form 2534-(R), 2 Feb 60

CODING FORM

PROBLEM Sample Problem

CODER DATE PAGE

LOCATION 1–6	ORDER TYPE 7–10	FORMULA STATEMENTS A 11–20 B 21–30 C 31–40	COMMENTS 41–80
		Statement of the Problem: Read an indefinite number of data points (X,Y) and determine the	
		least squares quadratic fit of the form $Y = C1 + C2X + C3X^2$, Assume X is in cols. 21-30	
		with an assumed point between cols. 24 and 25 and no exponent. Assume Y is in cols. 61-70	
		with an assumed point between cols. 66 and 67 and no exponent. Assume an integer code no.	
		is in cols. 75-80. Assume two blank cards follow each set of data. Print the coefficients	
		and compute the residuals. Print any residual that is larger than 2 in absolute value along	
		with its code number. Print three fl. pt. residuals per card and insert a blank card between	
		sets of output.	
	PROB	C-906 L.W.C. AUG. 1962 QUADRATIC FIT	
	BLOC	(A1,1 - A3,4/SY.)(V1 - V4) C1 - C3) R - R5	
	LAST	(X)Y)CODE	
N.SET		ENTER(PRINT BLANK)%	
BEGIN		CLEAR(9)NOS.AT(A1,1)% SET(I=0)%% MUST CLEAR MATRIX FOR F.N.E.	
1.1		READ-FORMAT(F1)-(X,I)Y,I)CODE,I % IF(09 = 0)GOTO(FIT)%	
		V1 = 1 % V2 = X,I % V3 = X,I ** 2 % V4 = Y,I %	
		ENTER(F.N.E.)A1,1)3)V1 % INC(I = I + 3)GOTO(1.1)%	
FIT		ENTER(SY.SNE)A1,1)3)C1 % PRINT(3)NOS.AT(C1) < COEFFS.> %	
		SET(J = 0) L = 0 %	
6.1		R,L = Y,J - (C1 + X,J(C2 + C3 * X,J))%% R,L IS RESIDUAL	
		IF - ABS(R,L < = 2)GOTO(8.1)% R1,L = CODE,J %	
		COUNT(6/2)IN(L)GOTO(8.1)	
		PRINT-FORMAT(P)-(6)NOS.AT(R)% SET(L = 0)%% PRINT RESIDUALS	
8.1		COUNT(I)BY(3)IN(J)GOTO(6.1)%	
		IF(L = 0)GOTO(N.SET)% PRINT-FORMAT(P)-(L)NOS.AT(R)% GOTO(N.SET)%	
F1	FORM	(3 - 20) 11 - 3 - 10) 3 - 30) 11 - 5 - 10) 3 - 5) 4 - 5 %	INPUT FORMAT
P	FORM	(9 - 12) 4 - 9) 3 - 3) 1 - 3) 5 - 1 - 8 %	OUTPUT FORMAT
	END	GOTO(BEGIN)	

DISTRIBUTION LIST

No. of Copies	Organization
20	Commander Defense Documentation Center ATTN: TIPCR Cameron Station Alexandria, Virginia 22314
1	Director Advanced Research Projects Agency Department of Defense Washington, D. C. 20310
1	Commanding Officer U. S. Army Communications Agency The Pentagon Washington 25, D. C.
1	Director, National Security Agency ATTN: R/D 36, Chief, Engineering Research Division Fort George G. Meade, Maryland 20755
1	Director P. O. Box 1925 ATTN: James Casey Washington, D. C. 20505
1	Commanding General U. S. Army Materiel Command ATTN: AMCRD-RP-B Washington, D. C. 20315
1	Commanding Officer Land Locomotion Laboratory Detroit Arsenal Warren, Michigan 48090
1	Commanding Officer Frankford Arsenal Philadelphia, Pennsylvania 19137
2	Commanding Officer Picatinny Arsenal ATTN: SNUPA-DR, Mr. S. Kravitz Artillery Ammunition Laboratory, Bldg 351 Feltman Research Lab Dover, New Jersey 07801
1	Commanding Officer Watertown Arsenal Watertown, Massachusetts 02172
1	Commanding Officer Harry Diamond Laboratories Washington, D. C. 20438
1	Commanding General U. S. Army Munitions Command Dover, New Jersey 07801
2	Commanding General U. S. Army Missile Command ATTN: Deputy Commanding General for Guided Missiles (1 cy) Redstone Arsenal, Alabama
1	Commanding General U. S. Army Weapons Command Rock Island, Illinois 61200
1	Commanding General White Sands Missile Range New Mexico 88002
1	Commanding General U. S. Army Chemical Corps Research and Development Command Washington 25, D. C.

DISTRIBUTION LIST

No. of Copies	Organization
1	Commanding Officer U. S. Army Chemical Warfare Laboratories ATTN: Dr. Carl M. Herget Edgewood Arsenal, Maryland 21040
1	Commanding Officer U. S. Army CBR Combat Development Agency Edgewood Arsenal, Maryland 21010
1	Commanding Officer U. S. Army Nuclear Defense Laboratory Edgewood Arsenal, Maryland 21010
1	Commanding Officer U. S. Army Operations Research Group Edgewood Arsenal, Maryland 21010
1	Commanding Officer U. S. Army Chemical Research and Development Labs Edgewood Arsenal, Maryland 21010
1	Commanding Officer U. S. Army Biological Labs Fort Detrick, Maryland 21701
1	Commanding General U. S. Army Chemical Corps Proving Grounds Dugway Proving Ground Dugway, Utah 84022
1	Commanding Officer U. S. Army Corps of Engineers Army Reactors Group Fort Belvoir, Virginia 22060
1	Commanding Officer U. S. Army Corps of Engineers Army Reactors Group Germantown, Maryland
1	Chief of Engineers Building T-7 Washington 25, D. C.
1	Director Geodesy, Intelligence and Mapping Research and Development Agency Fort Belvoir, Virginia 22060
1	Director Waterways Experiment Station P. O. Box 631 Vicksburg, Mississippi
2	Commanding General U. S. Army Engineering Research and Development Laboratories ATTN: STINFO Branch (1 cy) Fort Belvoir, Virginia 22060
1	Commanding General U. S. Army Cold Region Research and Engineering Laboratories P. O. Box 282 Hanover, New Hampshire 03755
1	Commanding General U. S. Army Medical Research and Development Command Washington 25, D. C.
1	Commanding Officer U. S. Army Medical Unit Fort Detrick, Maryland 21701
1	Director U. S. Army Medical Research Laboratory Fort Knox, Kentucky

DISTRIBUTION LIST

No. of Copies	Organization
1	Director U. S. Army Medical Research and Nutrition Laboratory Denver, Colorado
1	Commanding General U. S. Army Signal Missile Support Agency White Sands Missile Range New Mexico 88002
1	Commanding Officer U. S. Army Signal Electronic Research Unit P. O. Box 205 Mountain View, California
1	Commanding Officer U. S. Army Signal Avionics Field Office P. O. Box 209 St. Louis 66, Missouri
1	Commanding Officer U. S. Army Signal Engineering Agency Arlington Hall Station Arlington, Virginia
1	Commanding General U. S. Army Electronics Command ATTN: AMSEL-CB Fort Monmouth, New Jersey 07703
1	Commanding Officer U. S. Army Electronics Research and Development Laboratory ATTN: Data Equipment Branch Fort Monmouth, New Jersey 07703
1	Commanding General U. S. Army Electronics Proving Ground Fort Huachuca, Arizona 85613
1	Commanding Officer U. S. Transporation Materiel Command 12th and Spruce Streets St. Louis 66, Missouri
1	Commanding Officer U. S. Army Transporation Research Command Fort Eustis, Virginia 23604
1	Commanding General U. S. Combat Developments Command Fort Belvoir, Virginia 22060
1	Commanding General U. S. Continental Army Command Fort Monroe, Virginia 23351
1	Commandant U. S. Army Artillery & Guided Missile School Fort Sill, Oklahoma 73503
1	Commandant U. S. Army Guided Missile School Redstone Arsenal, Alabama 35809
1	Commandant U. S. Army Signal Corps School ATTN: Officer Department Fort Monmouth, New Jersey

DISTRIBUTION LIST

No. of Copies	Organization
1	Army Research Office 3045 Columbia Pike Arlington, Virginia 22204
1	Commanding Officer Army Research Office (Durham) Box CM, Duke Station Durham, North Carolina 27706
1	Commandant U. S. Army Command and General Staff College ATTN: Computing Facility Fort Leavenworth, Kansas 66027
1	Superintendent U. S. Military Academy Professor of Ordnance West Point, New York 10996
1	Commanding General ATTN: Computing Facility Fort George G. Meade, Maryland
1	Commanding Officer U. S. Army Major Item Data Agency Letterkenny Army Depot Chambersburg, Pennsylvania 17201
1	Contracting Officer Charlotte Ordnance Missile Plant 1820 Statesville Avenue Charlotte, North Carolina 28206
3	Chief, Bureau of Naval Weapons ATTN: DLI-3 Department of the Navy Washington, D. C. 20360
2	Commander U. S. Navy Ordnance Laboratory ATTN: Library White Oak Silver Spring, Maryland 20910
1	Commanding Officer U. S. Naval Ordnance Laboratory Corona, California 91720
1	Commander U. S. Naval Ordnance Test Station China Lake, California 93357
1	Superintendent U. S. Naval Postgraduate School ATTN: Technical Reports Section Monterey, California 93900
1	Director U. S. Naval Research Laboratory ATTN: Mr. Nassetta Washington, D. C. 20390
1	Commander U. S. Naval Weapons Laboratory ATTN: Computation & Analysis Branch Dahlgren, Virginia 22448
1	Chief of Naval Research Department of the Navy Washington, D. C.
1	Chief of Naval Operations Department of the Navy Washington, D. C. 20360
1	Commanding Officer and Director David W. Taylor Model Basin Washington, D. C. 20007

DISTRIBUTION LIST

No. of Copies	Organization
1	Chief, Bureau of Yards & Docks ATTN: Data Processing and Analysis Branch Department of the Navy Washington, D. C. 20360
1	Chief, Bureau of Ships ATTN: Computing Facility Department of the Navy Washington, D. C. 20360
1	Superintendent U. S. Naval Academy ATTN: Weapons Department Annapolis, Maryland
1	Commanding Officer U. S. Naval Air Development Center Johnsville, Pennsylvania 18974
1	Commanding Officer U. S. Naval Air Test Center ATTN: Armament Test Patuxent River, Maryland 20670
2	Commander U. S. Naval Missile Center ATTN: Simulation Branch Systems Department Range Operations Department Code 3280 Point Mugu, California 93041
1	Commanding Officer U. S. Naval Radiological Defense Laboratory San Francisco, California 94135
1	Director U. S. Naval Supersonic Lab Massachusetts Institute of Technology ATTN: Computer Facility 560 Memorial Drive Cambridge, Massachusetts 02139
1	Commander U. S. Naval Engineering Experiment Station ATTN: Applied Math Office, Code 502 Annapolis, Maryland
1	Commanding Officer Fleet Operations Control Center U. S. Pacific Fleet F. N. Quinn Navy No. 509 San Francisco FPO, California
1	FJSRL, OAR USAF Academy Colorado 80840
1	AEDC Arnold AFB Tennessee
1	Hq Comd USA (CDC - J. F. Cunningham) Bolling AFB Washington, D. C. 20332
1	AFFTC (FTTSD) Edwards AFB California 93523
1	TAWC (OA) Eglin AFB Florida 32542

DISTRIBUTION LIST

No. of Copies	Organization
1	AFCRL L. G. Hanscom Field Bedford, Massachusetts 01731
1	AFMDC (MDCS) Holloman AFB New Mexico 88330
1	TAC (OA) Langley AFB Virginia 23365
1	AUL (3T-AUL-60-118) Maxwell AFB Alabama 36112
1	ASD (Digital Computation Branch) Wright-Patterson AFB Ohio 45433
1	AFIT (MCLI) Wright-Patterson AFB Ohio 45433
1	CLIM Cen USAF Annex 2 225 D Street, S. E. Washington, D. C. 20333
1	Hq, USAF (AFAAC) Washington, D. C. 20330
2	Hq, USAF (AFNIN3) Washington, D. C. 20330
1	Director National Aeronautics and Space Administration 1520 H Street, N. W. Washington 25, D. C.
1	Director National Aeronautics and Space Administration Lewis Research Center ATTN: Computer Facility Cleveland Airport Cleveland, Ohio 44135
1	Director National Aeronautics and Space Administration Flight Research Center ATTN: Computer Facility Box 273 Edwards, California
2	Director National Aeronautics and Space Administration Goddard Space Flight Center ATTN: Tracking & Data Systems Computer Operations Br Data Systems Div Anacostia Naval Station 4555 Overlook Avenue, S.W. Washington 25, D. C.
1	U. S. Department of Commerce Bureau of Census ATTN: Computer Facility Federal Office Building No. 3 Suitland, Maryland
2	Director National Bureau of Standards National Applied Mathematics Laboratory ATTN: Miss Mary Stevens Dr. Franz L. Alt Computation Laboratory Washington 25, D. C.

DISTRIBUTION LIST

No. of Copies	Organization
2	Director National Bureau of Standards ATTN: Dr. S. N. Alexander Components & Techniques Section - Data Processing Systems Division 232 Dynamometer Building Washington, D. C. 20234
1	National Bureau of Standards Department of Commerce ATTN: William Youden Div 12 - 305CL Washington, D. C.
1	Aeronautical Chart and Information Center ATTN: Dominic P. Biagioli, ACCP Sam P. Scott, ACDEG-AO Second and Arsenal Street St. Louis, Missouri
1	Director Federal Aviation Agency National Aviation Facilities Experimental Station ATTN: Simulation and Computation Branch Atlantic City, New Jersey
1	Federal Aviation Agency ATTN: Data Processing Branch - Aircraft Management Division, Bureau of Flight Standards P. O. Box 1082 Oklahoma City, Oklahoma
1	Oak Ridge National Laboratory ATTN: Mr. E. C. Long P. O. Box X Oak Ridge, Tennessee 37831
1	Brookhaven National Laboratory ATTN: Computer Facility Upton, New York
1	Director Research Analysis Corporation ATTN: Computer Facility McLean, Virginia 22101
1	Applied Physics Laboratory ATTN: Computer Facility 8621 Georgia Avenue Silver Spring, Maryland 20910
1	American Data Processing Inc. ATTN: Allen Meacham 22 Floor, Book Tower Detroit 26, Michigan
1	Ampex Computer Products Co 9937 Jefferson Boulevard Culver City, California
1	E. I. DuPont De Nemours, Co Engineering Department ATTN: Theodore Baumeister, III Wilmington 98, Delaware
1	Engineering Research Associates Division of Remington Rand, Inc. 1902 W. Minnehaha Avenue St. Paul, Minnesota
1	Honneywell Incorporated ATTN: Mr. Donald M. Catton 1701 Pennsylvania Avenue Suite 804 Washington, D. C.
1	International Business Machines Corporation Engineering Laboratory ATTN: John Ashley - Customer Executive Education Department San Jose, California

DISTRIBUTION LIST

No. of Copies	Organization
1	Massey-Dickenson Company ATTN: Vicent Foxworth 151 Bearhill Road Waltman, Massachusetts
1	M-H Engineering and Research Center ATTN: Kenneth Curewitz 151 Needham Street Newton Highlands 61 Massachusetts
1	Raytheon Manufacturing Company P. O. Box 398 Bedford, Massachusetts 01730
1	Remington Rand Univac Division of Sperry Rand Corp. 1900 W. Allegheny Avenue St. Paul, Minnesota
1	Technitrol Engineering Corp. 1952 E. Alleghany Avenue Philadelphia 34, Pennsylvania
1	Watson Scientific Computing Laboratory 612 W. 116th Street New York 25, New York
1	California Institute of Technology ATTN: R. B. Gilmore - Comptroller Pasadena, California 91104
1	California Institute of Technology Jet Propulsion Laboratory ATTN: Computer Facility 4800 Oak Grove Drive Pasadena, California 91103
1	Columbia University Electronics Research Laboratory ATTN: G. S. Bodeen 632 West 125 Street New York 27, New York
1	Columbia University Lewis Cyclation Laboratory ATTN: Computer Facility Box 137 Irvington on Hudson, New York
1	Cornell University ATTN: John W. Hastie - Coordinator of Research Ithaca, New York 14850
1	Dartmouth College ATTN: Computation Center Hanover, New Hampshire
1	The George Washington University Logistics Research Project 707 22nd Street, N. W. Washington 7, D. C.
1	Georgia Institute of Technology Engineering Experiment Station ATTN: Rich Electronic Computer Center Atlanta 13, Georgia
1	Harvard University Computation Laboratory 33 Oxford Street Cambridge 38, Massachusetts
1	Indiana University ATTN: Research Computing Center Bloomington, Indiana

DISTRIBUTION LIST

No. of Copies	Organization
1	Iowa State University of Science and Technology Engineering Experiment Station ATTN: Robert M. Stewart, Jr. Cyclone Computer Lab Ames, Iowa
1	The Johns Hopkins University ATTN: Computation Center 34th and Charles Street Baltimore, Maryland 21218
1	Lehigh University ATTN: Computer Facility Bethlehem, Pennsylvania
1	Marquette University ATTN: Computing Center 1515 West Wisconsin Avenue Milwaukee, Wisconsin
1	Massachusetts Institute of Technology Lincoln Laboratory ATTN: Computer Facility Lexington 73, Massachusetts
1	Michigan State College College of Engineering ATTN: M. G. Kenney - Computing Laboratory East Lansing, Michigan
1	Missouri School of Mines and Metallurgy ATTN: Computer Facility Rolla, Missouri
1	New York University College of Engineering ATTN: Computation and Statistical Lab University Heights New York, New York 10053
1	Ohio State University Department of Industrial Engineering ATTN: Mr. Bander Columbus, Ohio
1	Oklahoma State University The Computing Center ATTN: D. R. Shreve - Department of Mathematics Stillwater, Oklahoma
1	Oregon State College Department of Mathematics ATTN: W. E. Milne Corvallis, Oregon
1	Polytechnic Institute of Brooklyn ATTN: Mr. Warren Boes 333 Jay Street Brooklyn, New York 11200
1	Princeton University Mathematics Department Princeton, New Jersey
1	Stanford University ATTN: Computation Center Stanford, California 94305
1	University of California ATTN: D. H. Lehmer 942 Hilldale Avenue Berkeley, California
1	University of Illinois Department of Mathematics ATTN: A. H. Taub Urbana, Illinois

DISTRIBUTION LIST

No. of Copies	Organization
1	University of Pennsylvania Moore School of Electrical Engineering Philadelphia 4, Pennsylvania
1	University of Wisconsin Numberical Analysis Department ATTN: P. J. Smith Madison, Wisconsin
1	Professor R. F. Jackson University of Delaware Newark, Delaware
1	Dr. Bansun Chang 12010 Susan Drive Granada Hills, California
1	Dr. Steven Lukasik Stevens Institute of Technology Davidson Laboratories Castle Point Station Hoboken, New Jersey
1	Dr. C. V. L. Smith U. S. Atomic Energy Commission Germantown, Maryland 20767
4	Australian Group c/o Military Attache Australian Embassy 2001 Connecticut Avenue, N.W. Washington, D. C. 20008
11	The Scientific Information Officer Defence Research Staff British Embassy 3100 Massachusetts Avenue, N.W. Washington, D. C. 20008
	Of Interest to: C-E-I-R Projects Office Turriff Building ATTN: Winston Riley III Great West Road Brentford, Middlesex England
4	Defense Research Member Canadian Joint Staff 2450 Massachusetts Avenue, N.W. Washington, D. C. 20008

Aberdeen Proving Ground

Chief, TIB

Air Force Liaison Office
Marine Corps Liaison Office
Navy Liaison Office
CDC Liaison Office

D & PS Branch Library

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Ballistic Research Laboratories Aberdeen Proving Ground, Md.	Unclassified
	2b GROUP

3. REPORT TITLE

THE FORAST PROGRAMMING LANGUAGE FOR ORDVAC AND BRLESC (REVISED)

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (Last name, first name, initial)

Campbell, Lloyd W. and Beck, Glenn A.

6 REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1965	154	3

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. 1P014501A14B	Report No. 1273
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. AVAILABILITY/LIMITATION NOTICES

Qualified requesters may obtain copies of this report from DDC.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	U. S. Army Materiel Command Washington, D. C.

13. ABSTRACT

FORAST is a procedure oriented programming language designed for use on the ORDVAC and BRLESC computers at BRL. Although it was designed for professional programmers, FORAST contains sufficient simple concepts to make it usable by a novice or journeyman. It permits the use of arithmetic formulas, some English word statements, and each computer accepts its own symbolic or absolute machine language. The latter feature permits the professional programmer to use the full power of each computer.

DD FORM 1 JAN 64 1473

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Programming Language Digital Computer BRLESC Computer ORDVAC Computer Compiler FORAST						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.